Joins, Split-Apply-Combine & MCPs

Abhijit Dasgupta

October 24, 2018

Goals today

- Learn how to join data sets (merging)
- Split-apply-combine
    - Split a dataset into a list of several datasets
    - Do something to each dataset
    - Put the results back together
- Use it for
    - Running tests for many variables
- Understand why we need multiple comparison procedures (MCP)
    - Things to think about

Data

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

- Clinical data: CSV|XLSX
- Proteome data: CSV|XLSX

Joins

Putting data sets together

- Quite often, data on individuals lie in different tables
    - Clinical, demographic and bioinformatic data

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)

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

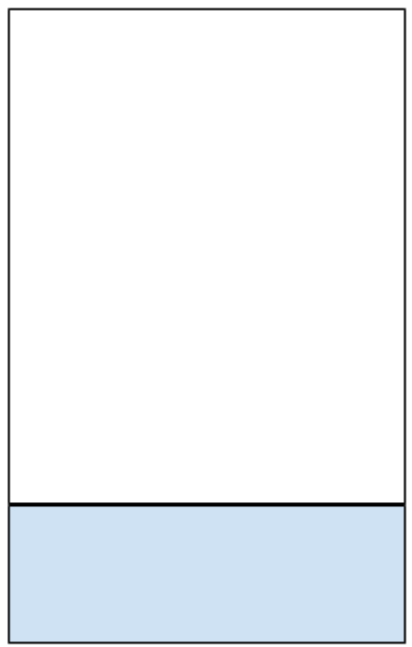
Joining data sets

We already talked about `cbind` and `rbind`:

`cbind`

Add columns

`rbind`



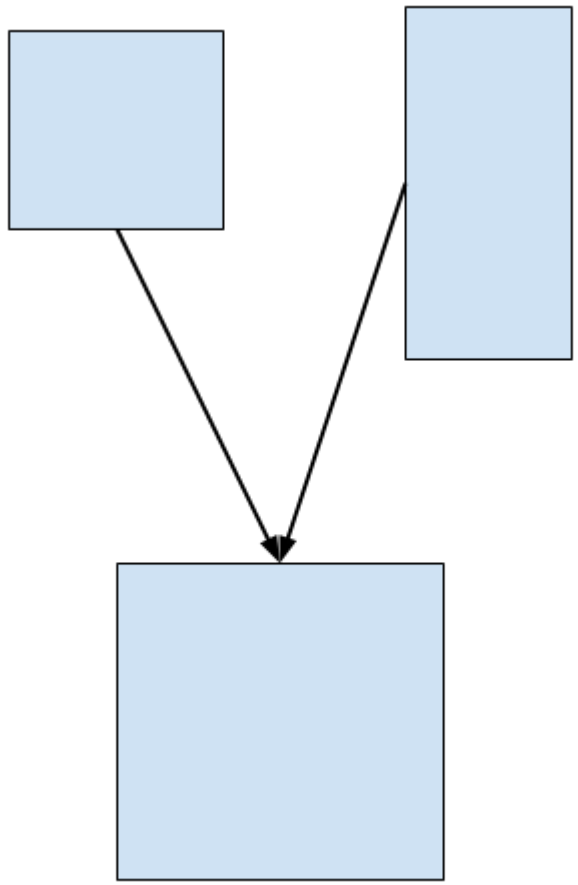
Add rows

Joining data sets

We will talk about more general ways of joining two datasets

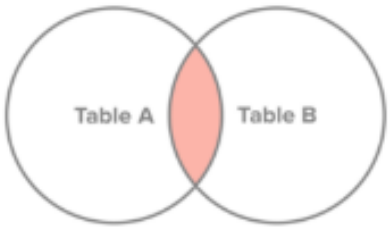
We will assume:

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
    - ID number
    - SSN (Social Security number)
    - Identifiable information



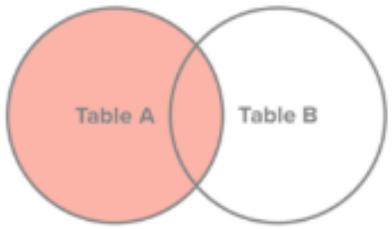
Joining data sets

Inner Join



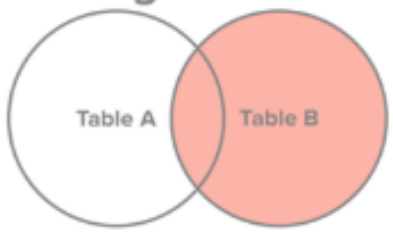
Select all records from Table A and Table B, where the join condition is met.

Left Join



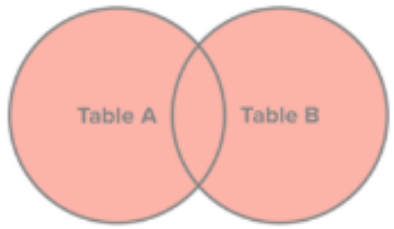
Select all records from Table A, along with records from Table B for which the join condition is met (if at all).

Right Join



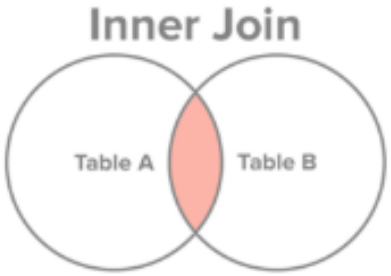
Select all records from Table B, along with records from Table A for which the join condition is met (if at all).

Full Join

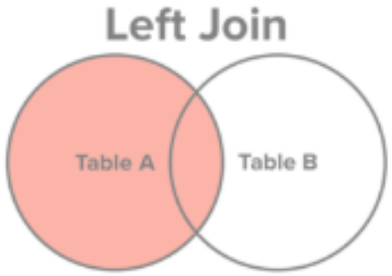


Select all records from Table A and Table B, regardless of whether the join condition is met or not.

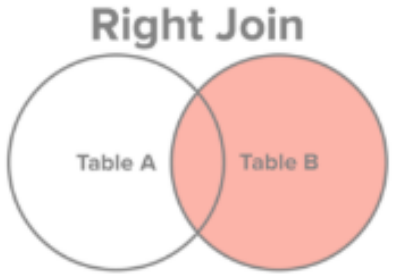
Joining data sets



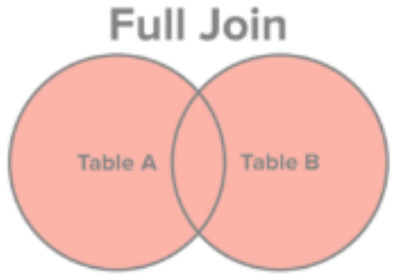
Select all records from Table A and Table B, where the join condition is met.



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



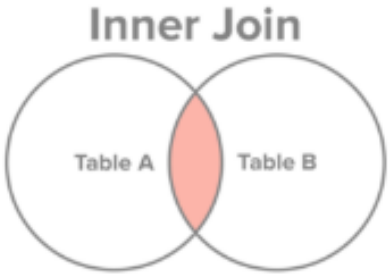
Select all records from Table B, along with records from Table A for which the join condition is met (if at all).



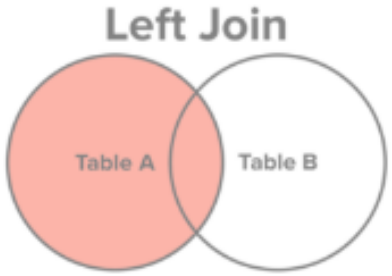
Select all records from Table A and Table B, regardless of whether the join condition is met or not.

inner_join	left_join	right_join	outer_join

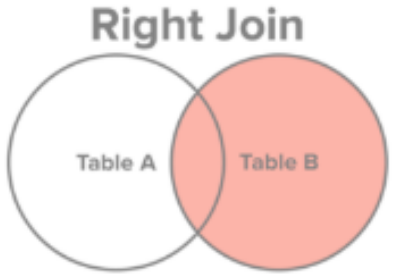
Joining data sets



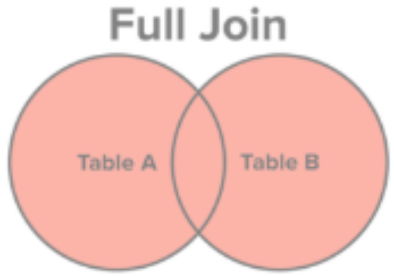
Select all records from Table A and Table B, where the join condition is met.



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

Data example

```
library(readxl)
clinical <- read_excel('lecture_joins_sac_data/BreastCancer_Clinical.xlsx') %>%
  set_names(str_replace_all(names(.), '[ -]+', '_'))
proteome <- read_excel('lecture_joins_sac_data/BreastCancer_Expression.xlsx') %>%
  set_names(str_replace_all(names(.), '[ -]+', '_'))
```

```
# A tibble: 105 x 30
  Complete_TCGA_ID Gender Age_at_Initial_Pathologic_D
  <chr>            <chr>
1 TCGA-A2-A0T2     FEMALE
2 TCGA-A2-A0CM     FEMALE
3 TCGA-BH-A18V     FEMALE
  PR_Status HER2_Final_Status Tumor Tumor_T1_Coded No
  <chr>     <chr>             <chr> <chr>          <c
1 Negative  Negative          T3    T_Other        N3
2 Negative  Negative          T2    T_Other        N0
3 Negative  Negative          T2    T_Other        N1
  Metastasis Metastasis_Coded AJCC_Stage Converted_St
  <chr>      <chr>            <chr>      <chr>
1 M1         Positive         Stage IV   No_Conversio
2 M0         Negative         Stage IIA  Stage IIA
3 M0         Negative         Stage IIB  No_Conversio
  Survival_Data_Form Vital_Status Days_to_Date_of_Las
  <chr>              <chr>
1 followup           DECEASED
2 followup           DECEASED
3 enrollment         DECEASED
  Days_to_date_of_Death OS_event OS_Time PAM50_mRNA
```

```
# A tibble: 83 x 11
  TCGA_ID      NP_958782 NP_958785 NP_958786 NP_00043
  <chr>            <dbl>     <dbl>     <dbl>     <dbl
1 TCGA-AO-A12D     1.10      1.11      1.11      1.11
2 TCGA-C8-A131     2.61      2.65      2.65      2.65
3 TCGA-AO-A12B    -0.660    -0.649    -0.654    -0.63
  NP_958783 NP_958784 NP_112598 NP_001611
      <dbl>     <dbl>     <dbl>     <dbl>
1     1.11      1.11     -1.52      0.483
2     2.65      2.65      3.91     -1.05
3    -0.649    -0.649    -0.618     1.22
# ... with 80 more rows
```

Data example

```
library(readxl)
clinical <- read_excel('lecture_joins_sac_data/BreastCancer_Clinical.xlsx') %>%
  set_names(str_replace_all(names(.), '[ -]+', '_'))
proteome <- read_excel('lecture_joins_sac_data/BreastCancer_Expression.xlsx') %>%
  set_names(str_replace_all(names(.), '[ -]+', '_'))
```

```
# 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
# ... with 102 more rows
```

```
# 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
# ... with 80 more rows
```

Data example

```
library(readxl)
clinical <- read_excel('lecture_joins_sac_data/BreastCancer_Clinical.xlsx') %>%
  set_names(str_replace_all(names(.), '[ -]+', '_'))
proteome <- read_excel('lecture_joins_sac_data/BreastCancer_Expression.xlsx') %>%
  set_names(str_replace_all(names(.), '[ -]+', '_'))
```

```
# 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
# ... with 102 more rows
```

```
# 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
# ... with 80 more rows
```

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

Data 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
```

Data 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

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

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

```
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_Pathologic_Diagnosis 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
  PR_Status HER2_Final_Status NP_958782 NP_958785 NP_958786 NP_000436
  <chr>     <chr>                 <dbl>     <dbl>     <dbl>     <dbl>
1 Negative  Negative              0.683     0.694     0.698     0.687
2 Negative  Negative              0.195     0.215     0.215     0.205
3 Negative  Negative             -1.12     -1.12     -1.12     -1.13
  NP_958781 NP_958780 NP_958783 NP_958784 NP_112598 NP_001611
      <dbl>     <dbl>     <dbl>     <dbl>     <dbl>     <dbl>
1     0.687     0.698     0.698     0.698     -2.65    -0.984
2     0.215     0.215     0.215     0.215     -1.04    -0.517
3    -1.13     -1.12     -1.12     -1.12       2.24    -2.58
# ... with 74 more rows
```

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_Pathologic_Diagnosis 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
  PR_Status HER2_Final_Status NP_958782 NP_958785 NP_958786 NP_000436
  <chr>     <chr>                 <dbl>     <dbl>     <dbl>     <dbl>
1 Negative  Negative              0.683     0.694     0.698     0.687
2 Negative  Negative              0.195     0.215     0.215     0.205
3 Negative  Negative             -1.12     -1.12     -1.12     -1.13
  NP_958781 NP_958780 NP_958783 NP_958784 NP_112598 NP_001611
      <dbl>     <dbl>     <dbl>     <dbl>     <dbl>     <dbl>
1     0.687     0.698     0.698     0.698     -2.65    -0.984
2     0.215     0.215     0.215     0.215     -1.04    -0.517
3    -1.13     -1.12     -1.12     -1.12       2.24    -2.58
# ... with 74 more rows
```

Note that we have all the columns from both datasets, but only 77 rows, which is the common set of IDs from the two datasets

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_Pathologic_Diagnosis 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
  PR_Status HER2_Final_Status NP_958782 NP_958785 NP_958786 NP_000436
  <chr>     <chr>                 <dbl>     <dbl>     <dbl>     <dbl>
1 Negative  Negative              0.683     0.694     0.698     0.687
2 Negative  Negative              0.195     0.215     0.215     0.205
3 Negative  Negative             -1.12     -1.12     -1.12     -1.13
  NP_958781 NP_958780 NP_958783 NP_958784 NP_112598 NP_001611
      <dbl>     <dbl>     <dbl>     <dbl>     <dbl>     <dbl>
1     0.687     0.698     0.698     0.698     -2.65    -0.984
2     0.215     0.215     0.215     0.215     -1.04    -0.517
3    -1.13     -1.12     -1.12     -1.12       2.24    -2.58
# ... with 74 more rows
```

Note that we have all the columns from both datasets, but only 77 rows, which is the common set of IDs from the two datasets

> If you don't include the `by` option, R will attempt to match values of any columns with the same names

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 Age_at_Initial_Pathologic_Diagnosis ER_Status
  <chr>            <chr>                                <dbl> <chr>
1 TCGA-A2-A0T2     FEMALE                                  66 Negative
2 TCGA-A2-A0CM     FEMALE                                  40 Negative
3 TCGA-BH-A18V     FEMALE                                  48 Negative
  PR_Status HER2_Final_Status NP_958782 NP_958785 NP_958786 NP_000436
  <chr>     <chr>                 <dbl>     <dbl>     <dbl>     <dbl>
1 Negative  Negative             NA        NA        NA        NA
2 Negative  Negative              0.683     0.694     0.698     0.687
3 Negative  Negative             NA        NA        NA        NA
  NP_958781 NP_958780 NP_958783 NP_958784 NP_112598 NP_001611
      <dbl>     <dbl>     <dbl>     <dbl>     <dbl>     <dbl>
1    NA        NA        NA        NA        NA        NA
2     0.687     0.698     0.698     0.698    -2.65     -0.984
3    NA        NA        NA        NA        NA        NA
# ... with 102 more rows
```

We get 105 rows, which is all the rows of `clinical`, combined with the rows of `proteome` with common IDs. The rest of the rows get `NA` for the proteome columns.

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 Age_at_Initial_Pathologic_Diagnosis ER_Status
  <chr>            <chr>                                <dbl> <chr>
1 TCGA-AO-A12D     FEMALE                                  43 Negative
2 TCGA-C8-A131     FEMALE                                  82 Negative
3 TCGA-AO-A12B     FEMALE                                  63 Positive
  PR_Status HER2_Final_Status NP_958782 NP_958785 NP_958786 NP_000436
  <chr>     <chr>                 <dbl>     <dbl>     <dbl>     <dbl>
1 Negative  Positive              1.10      1.11      1.11      1.11
2 Negative  Negative              2.61      2.65      2.65      2.65
3 Positive  Negative             -0.660    -0.649    -0.654    -0.632
  NP_958781 NP_958780 NP_958783 NP_958784 NP_112598 NP_001611
      <dbl>     <dbl>     <dbl>     <dbl>     <dbl>     <dbl>
1     1.12      1.11      1.11      1.11     -1.52      0.483
2     2.65      2.65      2.65      2.65      3.91     -1.05
3    -0.640    -0.654    -0.649    -0.649    -0.618     1.22
# ... with 77 more rows
```

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 Age_at_Initial_Pathologic_Diagnosis ER_Status
  <chr>            <chr>                                <dbl> <chr>
1 TCGA-AO-A12D     FEMALE                                  43 Negative
2 TCGA-C8-A131     FEMALE                                  82 Negative
3 TCGA-AO-A12B     FEMALE                                  63 Positive
  PR_Status HER2_Final_Status NP_958782 NP_958785 NP_958786 NP_000436
  <chr>     <chr>                 <dbl>     <dbl>     <dbl>     <dbl>
1 Negative  Positive              1.10      1.11      1.11      1.11
2 Negative  Negative              2.61      2.65      2.65      2.65
3 Positive  Negative             -0.660    -0.649    -0.654    -0.632
  NP_958781 NP_958780 NP_958783 NP_958784 NP_112598 NP_001611
      <dbl>     <dbl>     <dbl>     <dbl>     <dbl>     <dbl>
1     1.12      1.11      1.11      1.11     -1.52      0.483
2     2.65      2.65      2.65      2.65      3.91     -1.05
3    -0.640    -0.654    -0.649    -0.649    -0.618     1.22
# ... with 77 more rows
```

Here we get 80 rows, which is all the rows of `proteome`, along with the rows of `clinical` with common IDs, but with the columns of `clinical` appearing first.

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 Age_at_Initial_Pathologic_Diagnosis ER_Status
  <chr>            <chr>                                <dbl> <chr>
1 TCGA-A2-A0T2     FEMALE                                  66 Negative
2 TCGA-A2-A0CM     FEMALE                                  40 Negative
3 TCGA-BH-A18V     FEMALE                                  48 Negative
  PR_Status HER2_Final_Status NP_958782 NP_958785 NP_958786 NP_000436
  <chr>     <chr>                 <dbl>     <dbl>     <dbl>     <dbl>
1 Negative  Negative             NA        NA        NA        NA
2 Negative  Negative              0.683     0.694     0.698     0.687
3 Negative  Negative             NA        NA        NA        NA
  NP_958781 NP_958780 NP_958783 NP_958784 NP_112598 NP_001611
      <dbl>     <dbl>     <dbl>     <dbl>     <dbl>     <dbl>
1    NA        NA        NA        NA        NA        NA
2     0.687     0.698     0.698     0.698    -2.65     -0.984
3    NA        NA        NA        NA        NA        NA
# ... with 105 more rows
```

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 Age_at_Initial_Pathologic_Diagnosis ER_Status
  <chr>            <chr>                                <dbl> <chr>
1 TCGA-A2-A0T2     FEMALE                                  66 Negative
2 TCGA-A2-A0CM     FEMALE                                  40 Negative
3 TCGA-BH-A18V     FEMALE                                  48 Negative
  PR_Status HER2_Final_Status NP_958782 NP_958785 NP_958786 NP_000436
  <chr>     <chr>                 <dbl>     <dbl>     <dbl>     <dbl>
1 Negative  Negative             NA        NA        NA        NA
2 Negative  Negative              0.683     0.694     0.698     0.687
3 Negative  Negative             NA        NA        NA        NA
  NP_958781 NP_958780 NP_958783 NP_958784 NP_112598 NP_001611
      <dbl>     <dbl>     <dbl>     <dbl>     <dbl>     <dbl>
1    NA        NA        NA        NA        NA        NA
2     0.687     0.698     0.698     0.698    -2.65     -0.984
3    NA        NA        NA        NA        NA        NA
# ... with 105 more rows
```

Here we obtain 108 rows and 16 columns. So we've expanded the data in both rows and columns, putting missing values in where needed.

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

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`.

Are there protein expression differences between ER +ve and ER -ve breast cancers

Create analytic dataset

```
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 ER_Status NP_958782
  <chr>                                          <dbl> <chr>         <dbl>
1 TCGA-A2-A0CM                                      40 Negative      0.683
2 TCGA-BH-A18Q                                      56 Negative      0.195
3 TCGA-A7-A0CE                                      57 Negative     -1.12
  NP_958785 NP_958786 NP_000436 NP_958781 NP_958780 NP_958783 NP_958784
      <dbl>     <dbl>     <dbl>     <dbl>     <dbl>     <dbl>     <dbl>
1     0.694     0.698     0.687     0.687     0.698     0.698     0.698
2     0.215     0.215     0.205     0.215     0.215     0.215     0.215
3    -1.12     -1.12     -1.13     -1.13     -1.12     -1.12     -1.12
  NP_112598 NP_001611
      <dbl>     <dbl>
1     -2.65    -0.984
2     -1.04    -0.517
3      2.24    -2.58
# ... with 72 more rows
```

Protein-specific analyses

We want to analyze each protein separately, while maintaining alignment with ER status and age.

Protein-specific analyses

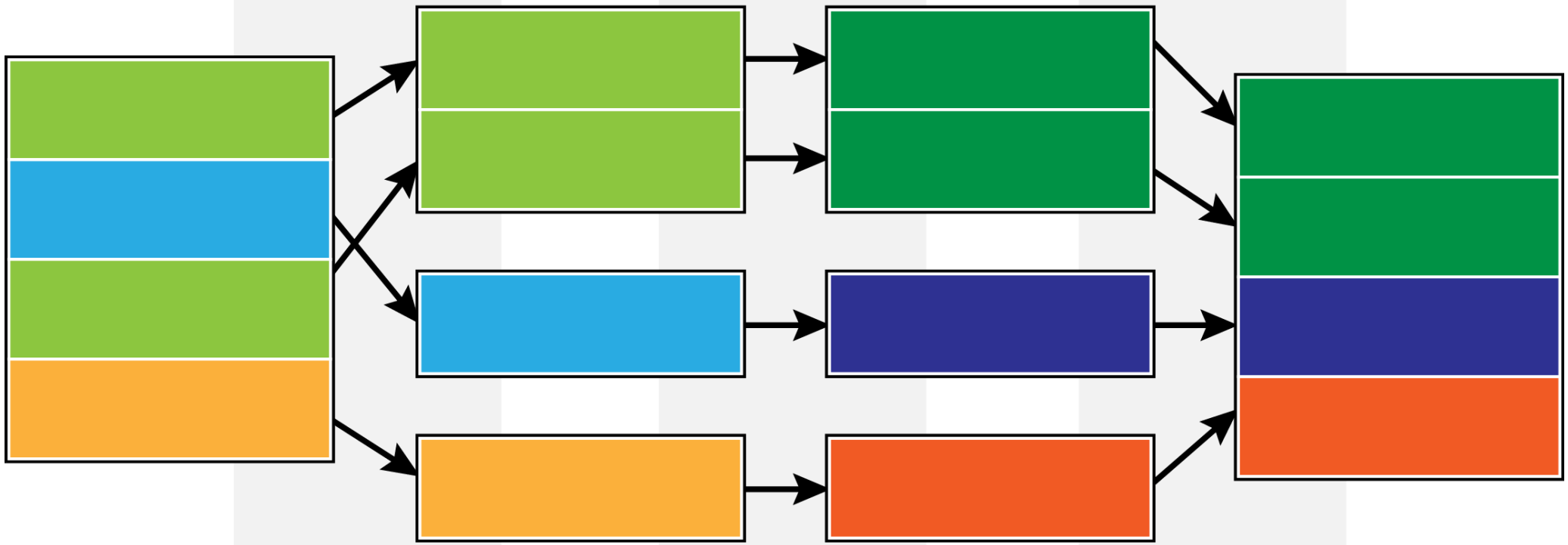
We want to analyze each protein separately, while maintaining alignment with ER status and age.

The R trick is to make this wide table long, so you can split on the rows

```
final_data2 <- final_data %>% gather(protein, expression, starts_with('NP')) %>%
  arrange(Complete_TCGA_ID)
```

```
# A tibble: 750 x 5
  Complete_TCGA_ID Age_at_Initial_Pathologic_Diagnosis ER_Status protein
  <chr>                                          <dbl> <chr>     <chr>
1 TCGA-A2-A0CM                                      40 Negative  NP_958782
2 TCGA-A2-A0CM                                      40 Negative  NP_958785
3 TCGA-A2-A0CM                                      40 Negative  NP_958786
  expression
       <dbl>
1      0.683
2      0.694
3      0.698
# ... with 747 more rows
```

Split-apply-combine



SPLIT
based on a common criteria

APPLY
a change to each group

COMBINE
the newly processed groups

Splitting data by protein

There are two ways of doing this:

```
final_data2_grp <- final_data2 %>% group_by(protein)
```

or

```
final_data2_nest <- final_data2 %>% nest(-protein)
```

Splitting data by protein

There are two ways of doing this:

```
final_data2_grp <- final_data2 %>% group_by(protein)
```

or

```
final_data2_nest <- final_data2 %>% nest(-protein)
```

```
# A tibble: 10 x 2
   protein    data
   <chr>      <list>
 1 NP_958782 <tibble [75 × 4]>
 2 NP_958785 <tibble [75 × 4]>
 3 NP_958786 <tibble [75 × 4]>
 4 NP_000436 <tibble [75 × 4]>
 5 NP_958781 <tibble [75 × 4]>
 6 NP_958780 <tibble [75 × 4]>
 7 NP_958783 <tibble [75 × 4]>
 8 NP_958784 <tibble [75 × 4]>
 9 NP_112598 <tibble [75 × 4]>
10 NP_001611 <tibble [75 × 4]>
```

Splitting data by protein

There are two ways of doing this:

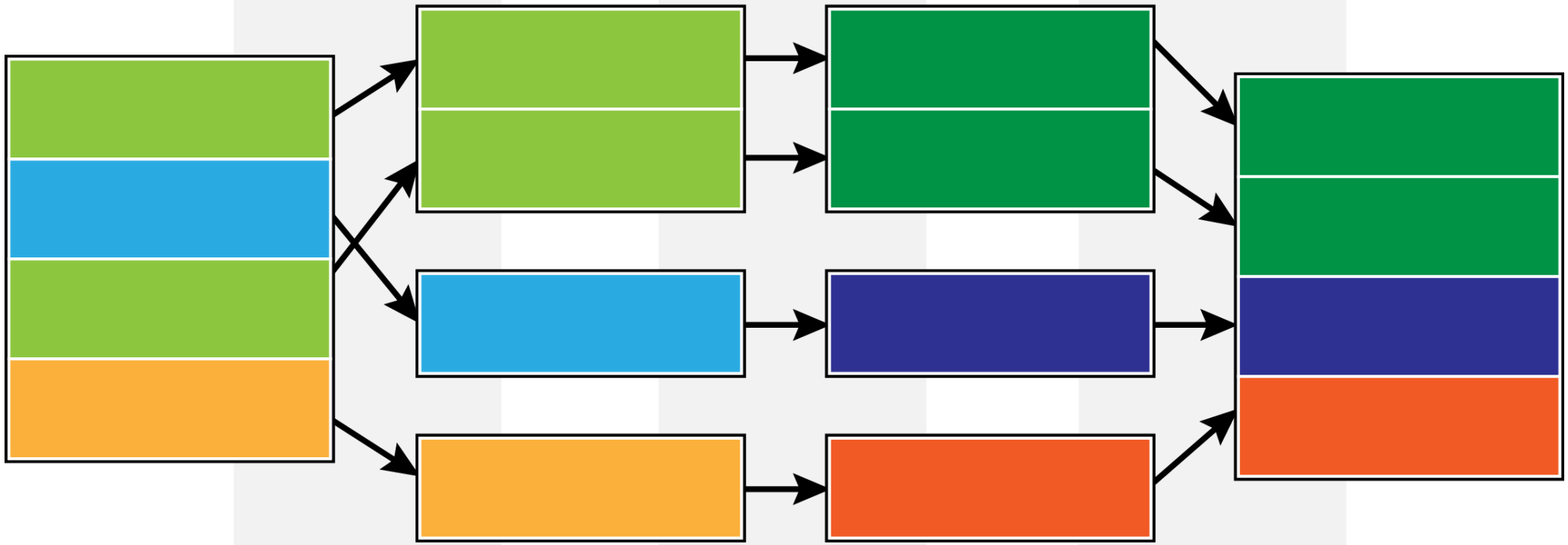
```
final_data2_grp <- final_data2 %>% group_by(protein)
```

or

```
final_data2_nest <- final_data2 %>% nest(-protein)
```

```
# A tibble: 10 x 2
   protein    data
   <chr>      <list>
 1 NP_958782 <tibble [75 × 4]>
 2 NP_958785 <tibble [75 × 4]>
 3 NP_958786 <tibble [75 × 4]>
 4 NP_000436 <tibble [75 × 4]>
 5 NP_958781 <tibble [75 × 4]>
 6 NP_958780 <tibble [75 × 4]>
 7 NP_958783 <tibble [75 × 4]>
 8 NP_958784 <tibble [75 × 4]>
 9 NP_112598 <tibble [75 × 4]>
10 NP_001611 <tibble [75 × 4]>
```

This is an example of a `list-column`. We will actually use this form, since it's a bit clearer to understand



SPLIT
based on a common criteria

APPLY
a change to each group

COMBINE
the newly processed groups

Side note: Functions

Functions

Functions are **rules** written in R code that take some *input* and give some *output*

```
#' @param d A data.frame object
#'
#' @return The p-value for the 2-sided t-test comparing expression between ER_Status
my_test <- function(d){
  ttest <- t.test(expression ~ ER_Status, data = d)
  return(ttest$p.value)
}
```

This function takes in a `data.frame`, does some operations on it (runs a t-test, and extracts the p-value) and returns a value (the p-value).

> In general, a function can output any kind of R object. We'll learn by example, but for more details, see this chapter in *R for Data Science* by Wickham & Grolemund.

We will **apply** this function to each split dataset in `final_data2_nest`

```
# A tibble: 10 x 2
   protein   data
   <chr>     <list>
 1 NP_958782 <tibble [75 × 4]>
 2 NP_958785 <tibble [75 × 4]>
 3 NP_958786 <tibble [75 × 4]>
 4 NP_000436 <tibble [75 × 4]>
 5 NP_958781 <tibble [75 × 4]>
 6 NP_958780 <tibble [75 × 4]>
 7 NP_958783 <tibble [75 × 4]>
 8 NP_958784 <tibble [75 × 4]>
 9 NP_112598 <tibble [75 × 4]>
10 NP_001611 <tibble [75 × 4]>
```

Let's take a look at an element in the `data` column

```
final_data2_nest$data[[1]]
```

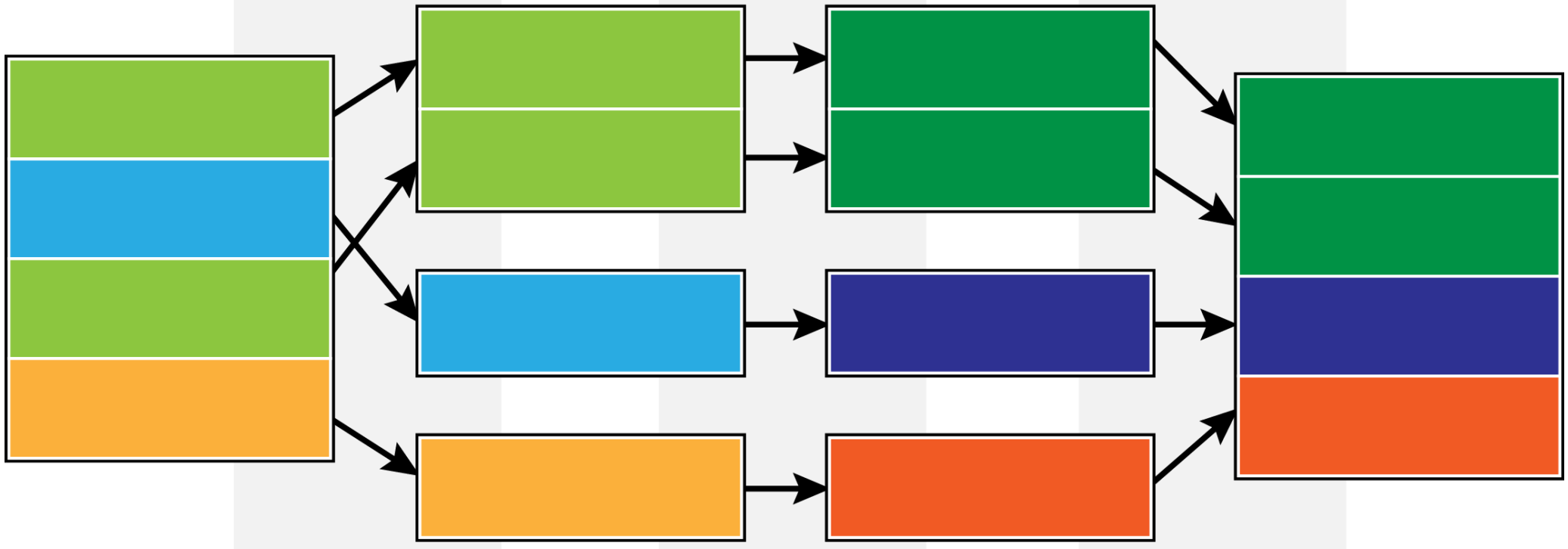
```
# A tibble: 75 x 4
  Complete_TCGA_ID Age_at_Initial_Pathologic_Diagnosis ER_Status
  <chr>                                          <dbl> <chr>
1 TCGA-A2-A0CM                                      40 Negative
2 TCGA-A2-A0D2                                      45 Negative
3 TCGA-A2-A0EQ                                      64 Negative
  expression
       <dbl>
1      0.683
2      0.107
3     -0.913
# ... with 72 more rows
```

Applying a function to each split dataset

We will use the function `purrr::map` to do this:

```
final_data2_nest %>% mutate(pval = map(data, my_test))
```

```
# A tibble: 10 x 3
   protein   data              pval
   <chr>     <list>            <list>
 1 NP_958782 <tibble [75 × 4]> <dbl [1]>
 2 NP_958785 <tibble [75 × 4]> <dbl [1]>
 3 NP_958786 <tibble [75 × 4]> <dbl [1]>
 4 NP_000436 <tibble [75 × 4]> <dbl [1]>
 5 NP_958781 <tibble [75 × 4]> <dbl [1]>
 6 NP_958780 <tibble [75 × 4]> <dbl [1]>
 7 NP_958783 <tibble [75 × 4]> <dbl [1]>
 8 NP_958784 <tibble [75 × 4]> <dbl [1]>
 9 NP_112598 <tibble [75 × 4]> <dbl [1]>
10 NP_001611 <tibble [75 × 4]> <dbl [1]>
```



SPLIT
based on a common criteria

APPLY
a change to each group

COMBINE
the newly processed groups

Combining the split results

```
final_data2_nest %>% mutate(pval = map(data, my_test)) %>%
  mutate(pval = unlist(pval))
```

```
# A tibble: 10 x 3
   protein    data                pval
   <chr>      <list>             <dbl>
 1 NP_958782 <tibble [75 × 4]> 0.924
 2 NP_958785 <tibble [75 × 4]> 0.934
 3 NP_958786 <tibble [75 × 4]> 0.940
 4 NP_000436 <tibble [75 × 4]> 0.922
 5 NP_958781 <tibble [75 × 4]> 0.926
 6 NP_958780 <tibble [75 × 4]> 0.930
 7 NP_958783 <tibble [75 × 4]> 0.931
 8 NP_958784 <tibble [75 × 4]> 0.931
 9 NP_112598 <tibble [75 × 4]> 0.908
10 NP_001611 <tibble [75 × 4]> 0.000432
```

Combining the split results

```
final_data2_nest %>% mutate(pval = map(data, my_test)) %>%
  mutate(pval = unlist(pval))
```

```
# A tibble: 10 x 3
   protein    data                pval
   <chr>      <list>             <dbl>
 1 NP_958782 <tibble [75 × 4]> 0.924
 2 NP_958785 <tibble [75 × 4]> 0.934
 3 NP_958786 <tibble [75 × 4]> 0.940
 4 NP_000436 <tibble [75 × 4]> 0.922
 5 NP_958781 <tibble [75 × 4]> 0.926
 6 NP_958780 <tibble [75 × 4]> 0.930
 7 NP_958783 <tibble [75 × 4]> 0.931
 8 NP_958784 <tibble [75 × 4]> 0.931
 9 NP_112598 <tibble [75 × 4]> 0.908
10 NP_001611 <tibble [75 × 4]> 0.000432
```

This could be done in one operation, as well

```
final_data2_nest %>% mutate(pval = map_dbl(data, my_test))
```

What's map doing?

```
final_data2_nest %>% mutate(pval = map_dbl(data, my_test)) %>% head(3)
```

```
# A tibble: 3 x 3
  protein   data               pval
  <chr>     <list>            <dbl>
1 NP_958782 <tibble [75 × 4]> 0.924
2 NP_958785 <tibble [75 × 4]> 0.934
3 NP_958786 <tibble [75 × 4]> 0.940
```

What's map doing?

```
final_data2_nest %>% mutate(pval = map_dbl(data, my_test)) %>% head(3)
```

```
# A tibble: 3 x 3
  protein   data               pval
  <chr>     <list>            <dbl>
1 NP_958782 <tibble [75 × 4]> 0.924
2 NP_958785 <tibble [75 × 4]> 0.934
3 NP_958786 <tibble [75 × 4]> 0.940
```

```
my_test(final_data2_nest$data[[1]])
```

```
[1] 0.9238144
```

What's map doing?

```
final_data2_nest %>% mutate(pval = map_dbl(data, my_test)) %>% head(3)
```

```
# A tibble: 3 x 3
  protein   data               pval
  <chr>     <list>            <dbl>
1 NP_958782 <tibble [75 × 4]> 0.924
2 NP_958785 <tibble [75 × 4]> 0.934
3 NP_958786 <tibble [75 × 4]> 0.940
```

```
my_test(final_data2_nest$data[[1]])
```

```
[1] 0.9238144
```

```
my_test(final_data2_nest$data[[2]])
```

```
[1] 0.9340165
```

The `map` function

1. `map(data, my_test)`:
    - `data` is a list of data.frames, and `my_test` is a function that takes a data.frame as input and produces some output, that is stored in a list
2. `map(data, ~t.test(expression ~ ER_Status, data = .))`:
    - Apply an anonymous function to each element of `data`, where the `.` serves as a place holder for an element of `data`. The anonymous function must start with a `~`. Note that the result of the anonymous function is the output of a `t.test`, which is a kind of object in R
3. `map(data, "ER_Status")`:
    - Extract the element `ER_Status` from each element of `data`

Pipelining this process

```
final_data2 %>% nest(-protein) %>%
  mutate(test = map(data,  ~t.test(expression ~ ER_Status, data = .))) %>%
  mutate(pval = map_dbl(test, 'p.value')) %>% head(3)
```

```
# A tibble: 3 x 4
  protein   data              test         pval
  <chr>     <list>            <list>      <dbl>
1 NP_958782 <tibble [75 × 4]> <S3: htest> 0.924
2 NP_958785 <tibble [75 × 4]> <S3: htest> 0.934
3 NP_958786 <tibble [75 × 4]> <S3: htest> 0.940
```

Pipelining this process

```
final_data2 %>% nest(-protein) %>%
  mutate(test = map(data,  ~t.test(expression ~ ER_Status, data = .))) %>%
  mutate(pval = map_dbl(test, 'p.value')) %>% head(3)
```

```
# A tibble: 3 x 4
  protein   data              test         pval
  <chr>     <list>            <list>      <dbl>
1 NP_958782 <tibble [75 × 4]> <S3: htest> 0.924
2 NP_958785 <tibble [75 × 4]> <S3: htest> 0.934
3 NP_958786 <tibble [75 × 4]> <S3: htest> 0.940
```

Cleaning it up

```
final_data2 %>% nest(-protein) %>%
  mutate(test = map(data,  ~t.test(expression ~ ER_Status, data = .))) %>%
  mutate(pval = map_dbl(test, 'p.value')) %>%
  select(protein, pval) %>% head(3)
```

```
# A tibble: 3 x 2
  protein    pval
  <chr>     <dbl>
1 NP_958782 0.924
2 NP_958785 0.934
3 NP_958786 0.940
```

Multiple comparison procedures (MCP)

Why do we need it?

- Recall, in hypothesis tests, the Type I error (or false positive rate) is

$$\Pr(Reject\ H_0 | H_0\ is\ true)$$

  This is typically limited by the testing procedure to 5%.

  - For the more technically interested, this is from the *Neyman-Pearson lemma*

- There is always a chance we are wrong!!

Why do we need it?

Imagine your test is like a biased coin, with heads being "Reject H_0" and tails being "Do not reject H_0"

Now assume H_0 is true, and you're doing multiple tests using the same data

Number of tests	Coin tosses	Pr(at least one head)
1	1 toss	0.05
2	2 tosses	0.10
5	5 tosses	0.23
10	10 tosses	0.40
100	100 tosses	0.99
1,000,000	1 million tosses	1.00

Why do we need it?

Imagine your test is like a biased coin, with heads being "Reject H_0" and tails being "Do not reject H_0"

Now assume H_0 is true, and you're doing multiple tests using the same data

Number of tests	Coin tosses	Pr(at least one head)
1	1 toss	0.05
2	2 tosses	0.10
5	5 tosses	0.23
10	10 tosses	0.40
100	100 tosses	0.99
1,000,000	1 million tosses	1.00

This means, if you're doing 1 million tests (like, e.g., a GWAS), the chance that you get **at least one false positive** is practically 1, i.e. a sure shot.

This requires some kind of multiple comparisons adjustment so we don't make excessive errors and get fooled into thinking we have significant results

Bonferroni correction

If you have *n* tests using the same data, then make sure that the Type I error is $0.05/n$. This means that for 100 tests, we'd reject the null hypothesis if the p-value was less than 0.0005 rather than 0.05.

How does this help?

Number of tests	Nominal FP rate	Corrected FP rate
1	0.050	0.050
2	0.098	0.049
5	0.226	0.049
10	0.401	0.049
100	0.994	0.049
1000000	1.000	0.049

The FP rate is the probability of getting at least one false positive.

Bonferroni correction

If you have *n* tests using the same data, then make sure that the Type I error is $0.05/n$. This means that for 100 tests, we'd reject the null hypothesis if the p-value was less than 0.0005 rather than 0.05.

How does this help?

Number of tests	Nominal FP rate	Corrected FP rate
1	0.050	0.050
2	0.098	0.049
5	0.226	0.049
10	0.401	0.049
100	0.994	0.049
1000000	1.000	0.049

The FP rate is the probability of getting at least one false positive.

Realize that this is quite stringent, since we're not allowing any false positives. The Bonferroni correction is thus quite comservative.

Bonferroni correction

There is a price to pay for this stringency, in terms of

> Statistical Power = Pr(Reject H_0 | H_0 is FALSE)
>
> This is the chance that the test will reject the null when the null hypothesis is wrong. We would like this to be high, so that we can detect true dfferences. This is usually set at 80%

Number of tests	Nominal FP rate	Bonferroni-corrected FP rate	Statistical power
1	0.050	0.050	0.800
2	0.098	0.049	0.706
5	0.226	0.049	0.573
10	0.401	0.049	0.474
100	0.994	0.049	0.212
1000	1.000	0.049	0.076
10000	1.000	0.049	0.023
100000	1.000	0.049	0.006
1000000	1.000	0.049	0.001

False discovery rates (FDR)

The FDR is

> the expected proportion of false positives (incorrectly rejected null hypotheses)

So if we set our FDR threshold to 0.05 and we identify 100 positives (reject 100 tests), then on average 5 of those 100 will be false positives

Benjamini-Hochberg (BH) procedure

This controls for FDR by ordering the p-values from highest to lowest and then judges their significance on a sliding scale. The adjusted values here are often referred to as *q-values*.

False discovery rates (FDR)

The FDR is

> the expected proportion of false positives (incorrectly rejected null hypotheses)

So if we set our FDR threshold to 0.05 and we identify 100 positives (reject 100 tests), then on average 5 of those 100 will be false positives

Benjamini-Hochberg (BH) procedure

This controls for FDR by ordering the p-values from highest to lowest and then judges their significance on a sliding scale. The adjusted values here are often referred to as *q-values*.

FDR control procedures retain more statistical power than Bonferroni corrections. There are other variants of this like the Benjamini-Yeukitili (BY) procedure

An example

If we see 10 heads in a row, we can statistically test whether the coin is biased or not.

```
prop.test(x = 10, n = 10, p = 0.5) # x = heads, n = tosses
```

```

    1-sample proportions test with continuity correction

data:  10 out of 10, null probability 0.5
X-squared = 8.1, df = 1, p-value = 0.004427
alternative hypothesis: true p is not equal to 0.5
95 percent confidence interval:
 0.6554628 1.0000000
sample estimates:
p
1
```

P-value adjustment

Suppose we want to test if pennies in circulation are biased (i.e. chance of a head is not 0.5). We collect 20,000 pennies and flip each of then 100 times, recording the number of heads. We can simulate this experiment in R using the `rbinom` function, use the `prop.test` function to test our hypothesis, and then get adjusted p-values using the `p.adjust` function.

```
set.seed(1243) # Initialize the random number generator
coinTable <- tibble(heads = rbinom(n = 20000, size = 100, prob = 0.5))
coinTable <- coinTable %>% mutate(pvals = map_dbl(heads, ~prop.test(., 100, 0.5)$p.value))
coinTable <- coinTable %>% mutate(bonf = p.adjust(pvals, method = 'bonferroni')) %>%
   mutate(q_value = p.adjust(pvals, method = 'fdr'))
```

```
# A tibble: 20,000 x 4
  heads pvals  bonf q_value
  <int> <dbl> <dbl>   <dbl>
1    56 0.271     1   0.996
2    47 0.617     1   0.996
3    48 0.764     1   0.996
# ... with 2e+04 more rows
```

The proportion of p-values < 0.05 is:

```
# A tibble: 1 x 3
   pvals  bonf q_value
   <dbl> <dbl>   <dbl>
1 0.0356     0       0
```

P-value adjustment

If the coins are truly biased, we can see that the Bonferroni misses most of them whereas the q-value doesn't.

```
set.seed(1243) # Initialize the random number generator
coinTable <- tibble(heads = rbinom(n = 20000, size = 100, prob = 0.7))
coinTable <- coinTable %>% mutate(pvals = map_dbl(heads, ~prop.test(., 100, 0.5)$p.value))
coinTable <- coinTable %>% mutate(bonf = p.adjust(pvals, method = 'bonferroni')) %>%
  mutate(q_value = p.adjust(pvals, method = 'fdr'))
```

```
# A tibble: 20,000 x 4
  heads     pvals  bonf   q_value
  <int>     <dbl> <dbl>     <dbl>
1    64 0.00693   1     0.00753
2    72 0.0000171 0.342 0.0000451
3    71 0.0000413 0.826 0.0000892
# ... with 2e+04 more rows
```

```
coinTable %>% summarize_at(vars(-heads), funs(mean(. < 0.05)))
```

```
# A tibble: 1 x 3
  pvals  bonf q_value
  <dbl> <dbl>   <dbl>
1 0.979 0.168   0.979
```

Next week: Statistical models in R