Practical R

Abhijit Dasgupta

15 February, 2021

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 Canvas page and the expectation is that you will 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)

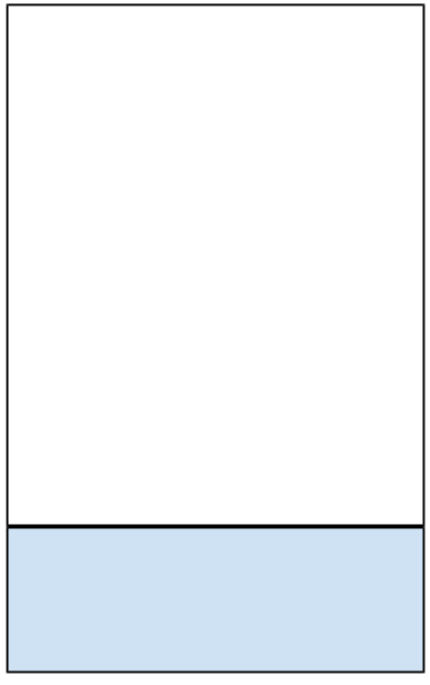
Joining data sets

`cbind(x,y)`

Add columns

Data sets have same subjects/observations, but new variables

`rbind(x,y)`



Add rows

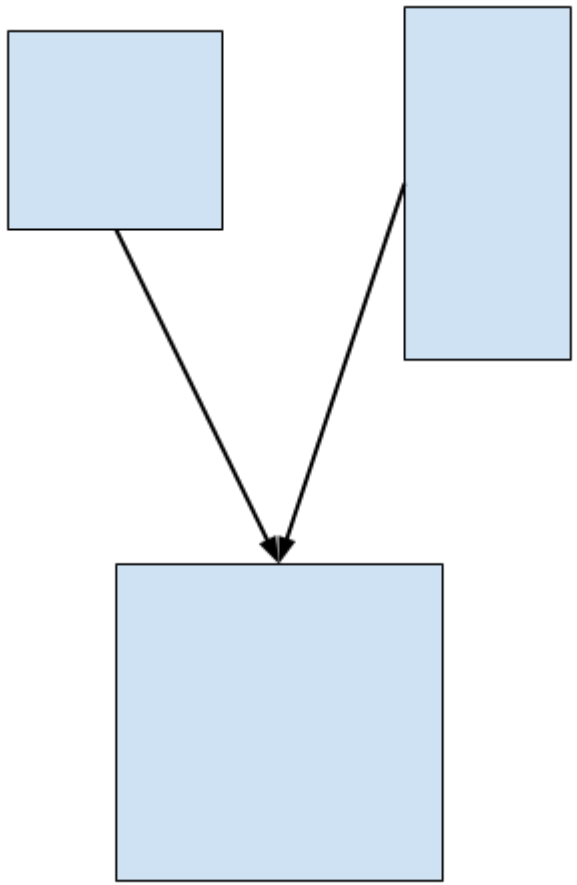
Data sets have same variables, but new subjects

Joining data sets

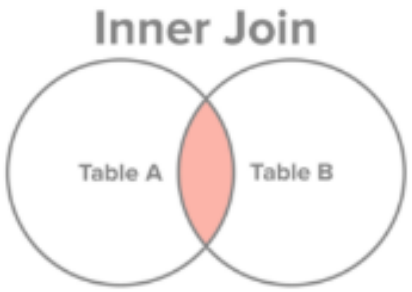
We will talk about more general ways of joining two datasets

We will assume:

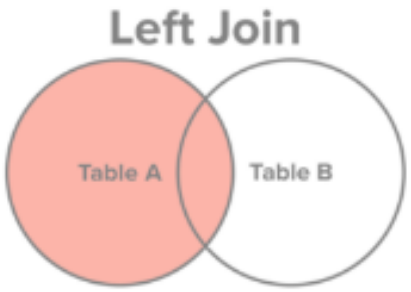
1. We have two rectangular data sets (so `data.frame` or `tibble`)
2. There is at least one variable (column) in common, even if they have different names
    - Patient ID number
    - SSN (Social Security number)
    - Identifiable information



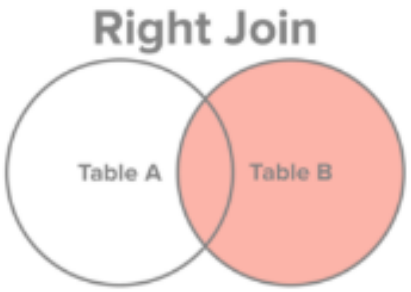
Joining data sets



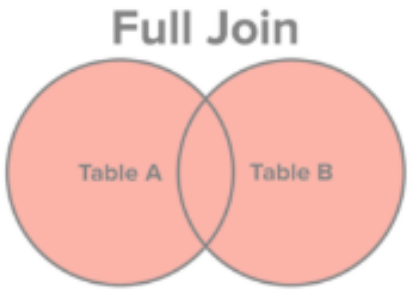
Select all records from Table A and Table B, where the join condition is met.



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') # See ?tibble::tibble
proteome <- read_excel('data/BreastCancer_Expression.xlsx',
                       .name_repair = 'universal')
```

```
clinical
```

```
# A tibble: 105 x 30
   Complete.TCGA.ID Gender Age.at.Initial.… ER
   <chr>            <chr>             <dbl> <c
 1 TCGA-A2-A0T2     FEMALE               66 Neg
 2 TCGA-A2-A0CM     FEMALE               40 Neg
 3 TCGA-BH-A18V     FEMALE               48 Neg
 4 TCGA-BH-A18Q     FEMALE               56 Neg
 5 TCGA-BH-A0E0     FEMALE               38 Neg
 6 TCGA-A7-A0CE     FEMALE               57 Neg
 7 TCGA-D8-A142     FEMALE               74 Neg
 8 TCGA-A2-A0D0     FEMALE               60 Neg
 9 TCGA-AO-A0J6     FEMALE               61 Neg
10 TCGA-A2-A0YM     FEMALE               67 Neg
# … with 95 more rows, and 24 more variables:
#   Tumor..T1.Coded <chr>, Node <chr>, Node.Co
```

```
proteome
```

```
# A tibble: 83 x 11
   TCGA_ID NP_958782 NP_958785 NP_958786 NP_00
   <chr>       <dbl>     <dbl>     <dbl>    <
 1 TCGA-A…      1.10      1.11      1.11      1
 2 TCGA-C…      2.61      2.65      2.65      2
 3 TCGA-A…     -0.660    -0.649    -0.654    -0
 4 TCGA-B…      0.195     0.215     0.215     0
 5 TCGA-C…     -0.494    -0.504    -0.501    -0
 6 TCGA-C…      2.77      2.78      2.78      2
 7 TCGA-E…      0.863     0.870     0.870     0
 8 TCGA-C…      1.41      1.41      1.41      1
 9 TCGA-A…      1.19      1.19      1.19      1
10 TCGA-A…      1.10      1.10      1.10      1
# … with 73 more rows, and 3 more variables: N
#   NP_001611 <dbl>
```

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
```

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

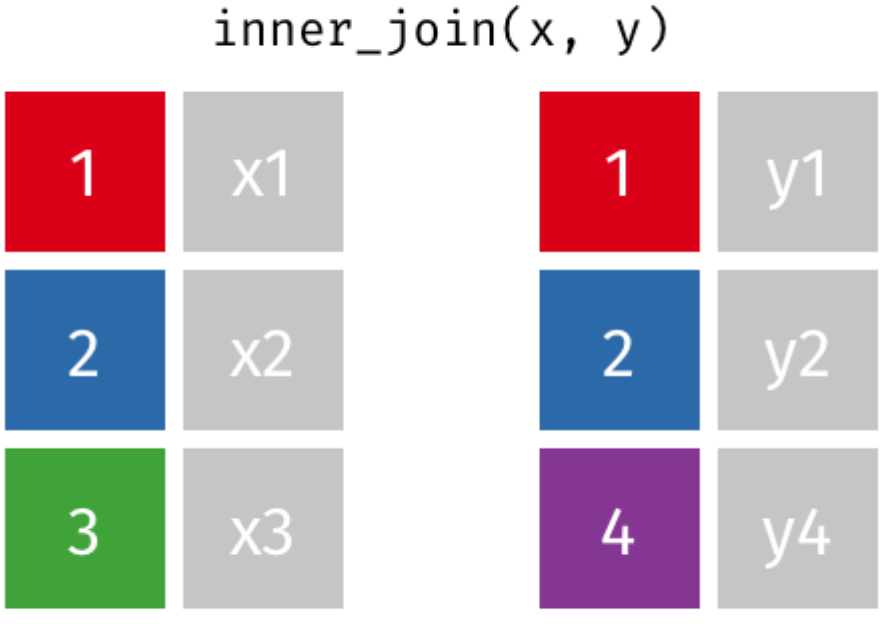
```
proteome <- proteome %>% filter(!duplicated(TCGA_ID))
```

> `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

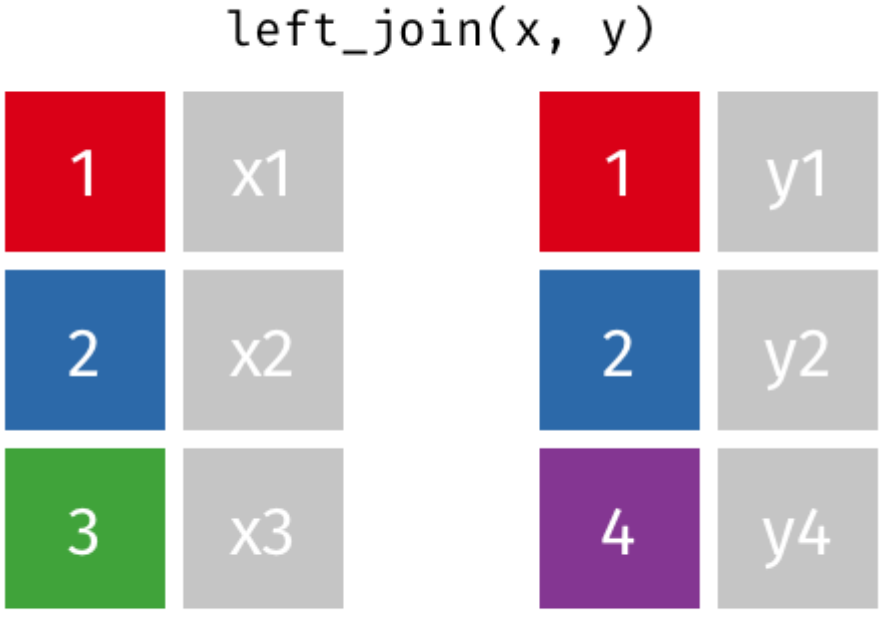
Inner join

```
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 PR.Status HER2.Final.Stat…
   <chr>            <chr>              <dbl> <chr>     <chr>     <chr>
 1 TCGA-A2-A0CM     FEMALE                40 Negative  Negative  Negative
 2 TCGA-BH-A18Q     FEMALE                56 Negative  Negative  Negative
 3 TCGA-A7-A0CE     FEMALE                57 Negative  Negative  Negative
 4 TCGA-D8-A142     FEMALE                74 Negative  Negative  Negative
 5 TCGA-AO-A0J6     FEMALE                61 Negative  Negative  Negative
 6 TCGA-A2-A0YM     FEMALE                67 Negative  Negative  Negative
 7 TCGA-A2-A0D2     FEMALE                45 Negative  Negative  Negative
 8 TCGA-A2-A0SX     FEMALE                48 Negative  Negative  Negative
 9 TCGA-AO-A0JL     FEMALE                59 Negative  Negative  Negative
10 TCGA-AO-A12F     FEMALE                36 Negative  Negative  Negative
# … with 67 more rows, and 10 more variables: 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.

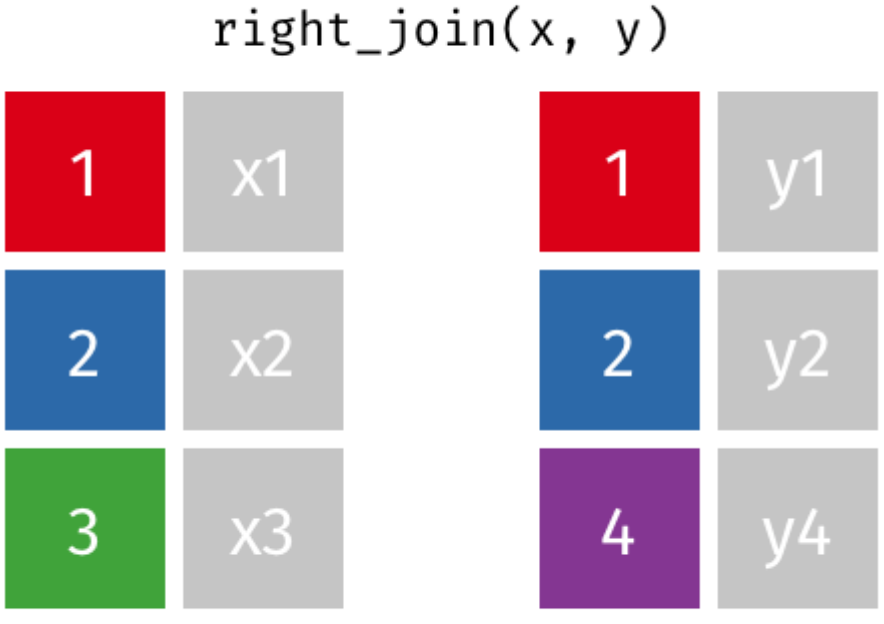
Left join

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

```
# A tibble: 105 x 16
  Complete.TCGA.ID Gender 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 NP_958781
  <chr>     <chr>                 <dbl>     <dbl>     <dbl>     <dbl>     <dbl>
1 Negative  Negative             NA        NA        NA        NA        NA
2 Negative  Negative              0.683     0.694     0.698     0.687     0.687
3 Negative  Negative             NA        NA        NA        NA        NA
  NP_958780 NP_958783 NP_958784 NP_112598 NP_001611
      <dbl>     <dbl>     <dbl>     <dbl>     <dbl>
1    NA        NA        NA        NA        NA
2     0.698     0.698     0.698    -2.65     -0.984
3    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



- 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

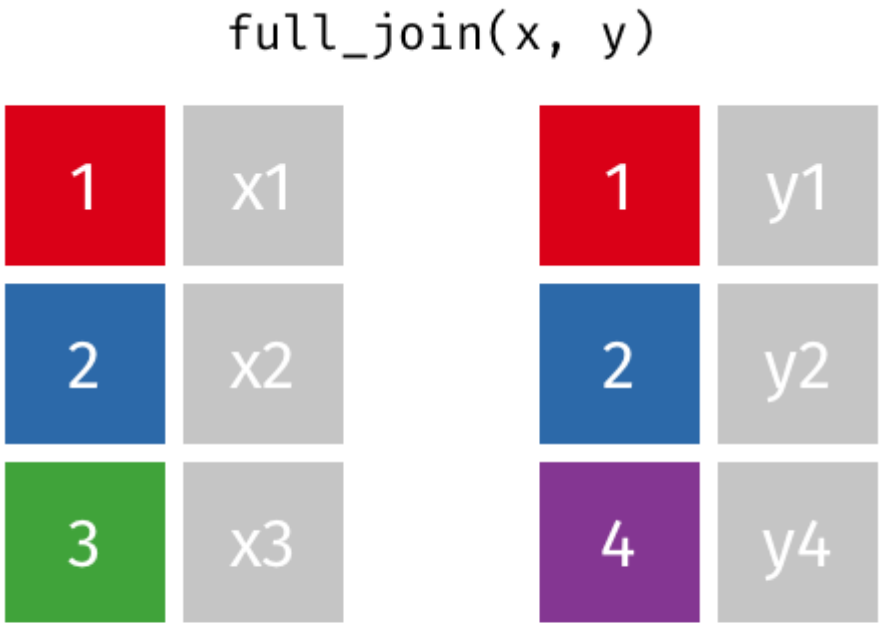
Right join

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

```
# A tibble: 80 x 16
  Complete.TCGA.ID Gender 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 NP_958781
  <chr>     <chr>                 <dbl>     <dbl>     <dbl>     <dbl>     <dbl>
1 Negative  Negative              0.683     0.694     0.698     0.687     0.687
2 Negative  Negative              0.195     0.215     0.215     0.205     0.215
3 Negative  Negative             -1.12     -1.12     -1.12     -1.13     -1.13
  NP_958780 NP_958783 NP_958784 NP_112598 NP_001611
      <dbl>     <dbl>     <dbl>     <dbl>     <dbl>
1     0.698     0.698     0.698     -2.65    -0.984
2     0.215     0.215     0.215     -1.04    -0.517
3    -1.12     -1.12     -1.12       2.24    -2.58
# … 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



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 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 NP_958781
  <chr>     <chr>                 <dbl>     <dbl>     <dbl>     <dbl>     <dbl>
1 Negative  Negative             NA        NA        NA        NA        NA
2 Negative  Negative              0.683     0.694     0.698     0.687     0.687
3 Negative  Negative             NA        NA        NA        NA        NA
  NP_958780 NP_958783 NP_958784 NP_112598 NP_001611
      <dbl>     <dbl>     <dbl>     <dbl>     <dbl>
1    NA        NA        NA        NA        NA
2     0.698     0.698     0.698    -2.65     -0.984
3    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`. 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 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
```

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 fi
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 comm
```

> 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

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.

Categorical variables

What are categorical variables?

Categorical variables are variables that

- have values defining categories of things
- typically have a few unique values
- may or may not be ordered
- are not interval-scaled, i.e., their differences don't make sense *per se*

What are categorical variables?

Non-ordered

1. Race (White, Black, Hispanic, Asian, Native American)
2. Gender (Male, Female, Other)
3. Geographic regions (Africa, Asia, Europe, North America, South America)
4. Genes/Proteins

Ordered

1. Income levels (< $10K, $10K - $25K, $25K - $75K, $75K - $100K)
2. BMI categories (Underweight, Normal, Overweight, Obese)
3. Number of bedrooms in houses (1 BR, 2BR, 3BR, 4BR)

Cateogical variables in R

The `factor` data type

R stores categorical variables as type `factor`.

- You can coerce a character or numeric object into a factor using `as.factor`.
- You can check if an object is a factor with `is.factor`.
- You can create a factor with the function `factor`.

The `factor` data type

> `factor(x = character(), levels, labels = levels, exclude = NA, ordered = is.ordered(x), nmax = NA)`
>
> factor returns an object of class "factor" which has a set of integer codes the length of x with a "levels" attribute of mode character and unique

---

- Internally, each level of a factor is coded as an integer
- Each such integer has a corresponding `level` which is a character, describing the level.
- You can add `labels` to each level to change the printed form of the factor.

The factor data type

```
x <- c('Maryland','Virginia', 'District', 'Maryland','Virginia')  # a character vector
xf <- as.factor(x)
xf
```

```
[1] Maryland Virginia District Maryland Virginia
Levels: District Maryland Virginia
```

There are three levels, that by default are in alphabetical order

---

```
as.integer(xf)
```

```
[1] 2 3 1 2 3
```

- District = 1, Maryland = 2, Virginia = 3

```
as.character(xf)
```

```
[1] "Maryland" "Virginia" "District" "Maryland
```

- Get original characters back

The factor data type

```
y <- c(5, 3, 9, 4, 5, 3)
yf <- as.factor(y)
yf
```

```
[1] 5 3 9 4 5 3
Levels: 3 4 5 9
```

Levels are still in alphanumeric order

```
as.numeric(yf)
```

```
[1] 3 1 4 2 3 1
```

- Note, we don't get original integers back!!
- 3 = 1, 4 = 2, 5 = 3, 9 = 4

```
as.numeric(as.character(yf))
```

```
[1] 5 3 9 4 5 3
```

- This is how you get numbers back

The factor data type

```
x <- c('MD','DC','VA','MD','DC')
xf <- factor(x)
unclass(xf)
```

```
[1] 2 1 3 2 1
attr(,"levels")
[1] "DC" "MD" "VA"
```

```
x <- c('MD','DC','VA','MD','DC')
xf <- factor(x, levels = c('MD','DC','VA'))
unclass(xf)
```

```
[1] 1 2 3 1 2
attr(,"levels")
[1] "MD" "DC" "VA"
```

- If I change the level designation, the underlying coding changes
- This is important when a factor is an independent variable in a regression model

The `factor` data type

The `drv` variable in the `mpg` dataset tells us the kind of drive (front, rear or 4-wheel) each car has. However it's coded as `f`, `r`, and `4`, which is not great for display purposes. We can re-label these levels, but we have to be a bit careful

```
x <- mpg$drv
xf <- factor(x,
             levels = c('4-wheel','Front whee
                        'Rear wheel'))
head(xf)
```

```
[1] <NA> <NA> <NA> <NA> <NA> <NA>
Levels: 4-wheel Front wheel Rear wheel
```

```
x <- mpg$drv
xf <- factor(x,
             levels = c('4', 'f','r'),
             labels = c('4-wheel','Front whee
                        'Rear wheel'))
head(xf)
```

```
[1] Front wheel Front wheel Front wheel Front w
Levels: 4-wheel Front wheel Rear wheel
```

Levels have to match what's actually in the original data, but you can re-label the levels.

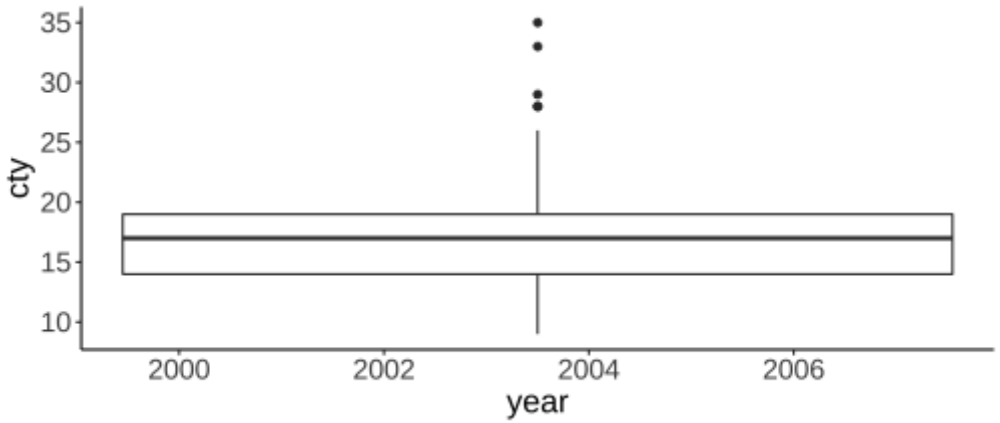
Why factors?

Factors are R's discrete data type

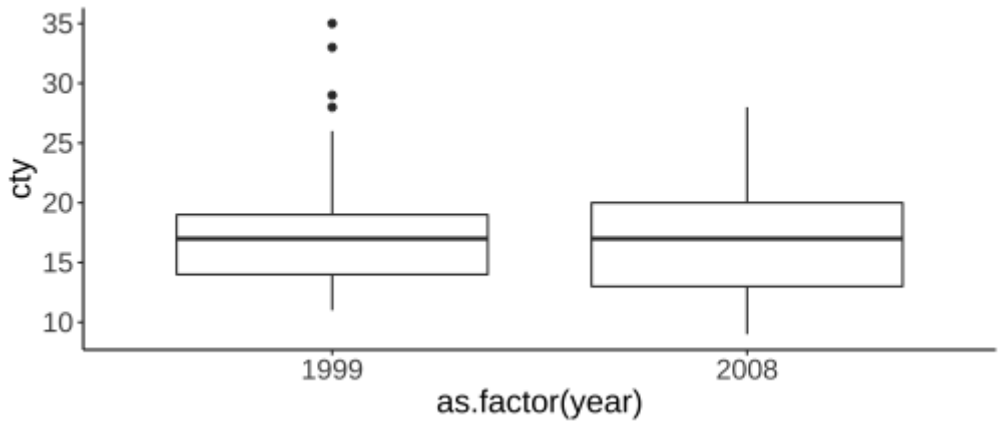
- Factors are interpreted as discrete by R's functions

```
ggplot(mpg,
       aes(year, cty))+
  geom_boxplot()
```

```
Warning: Continuous x aesthetic -- did you forg
```




```
ggplot(mpg,
       aes(as.factor(year), cty))+
  geom_boxplot()
```



Dummy variables are automatically created from factors

```
model.matrix(~species, data = palmerpenguins::
```

```
  (Intercept) speciesChinstrap speciesGentoo
1           1                0             0
2           1                0             0
3           1                1             0
4           1                1             0
5           1                0             1
6           1                0             1
attr(,"assign")
[1] 0 1 1
attr(,"contrasts")
attr(,"contrasts")$species
[1] "contr.treatment"
```

- If a factor has *n* levels, you get *n-1* dummy variables
- The level corresponding to integer code 1 is omitted as the reference level

---

Changing the base level (integer code 1) changes model interpretation since it changes the reference level against which all other levels are compared.

Manipulating factors

The `forcats` package (part of `tidyverse`)

Effect in models

```
library(palmerpenguins)
m <- lm(body_mass_g ~ species, data = penguins
broom::tidy(m)
```

```
# A tibble: 3 x 5
  term             estimate std.error statisti
  <chr>               <dbl>     <dbl>     <dbl
1 (Intercept)        3701.       37.6    98.4
2 speciesChinstrap     32.4      67.5     0.48
3 speciesGentoo      1375.       56.1    24.5
```

Compare with Adele

```
p1 <- penguins %>%
  mutate(species = fct_relevel(species, 'Gento
m1 <- lm(body_mass_g ~ species, data=p1 )
broom::tidy(m1)
```

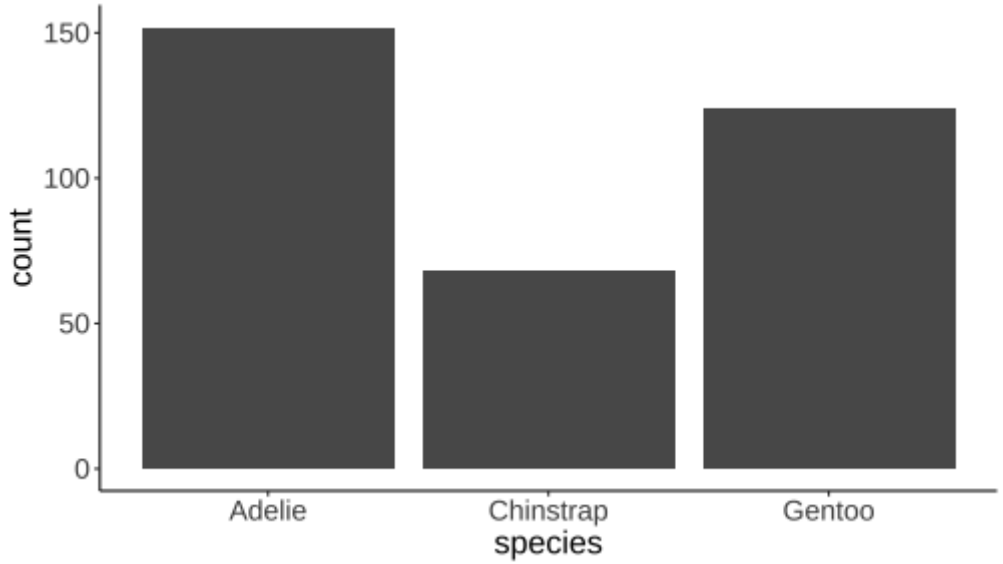
```
# A tibble: 3 x 5
  term             estimate std.error statisti
  <chr>               <dbl>     <dbl>     <dbl
1 (Intercept)        5076.       41.7     122.
2 speciesAdelie     -1375.       56.1     -24.
3 speciesChinstrap  -1343.       69.9     -19.
```

Compare with Gentoo

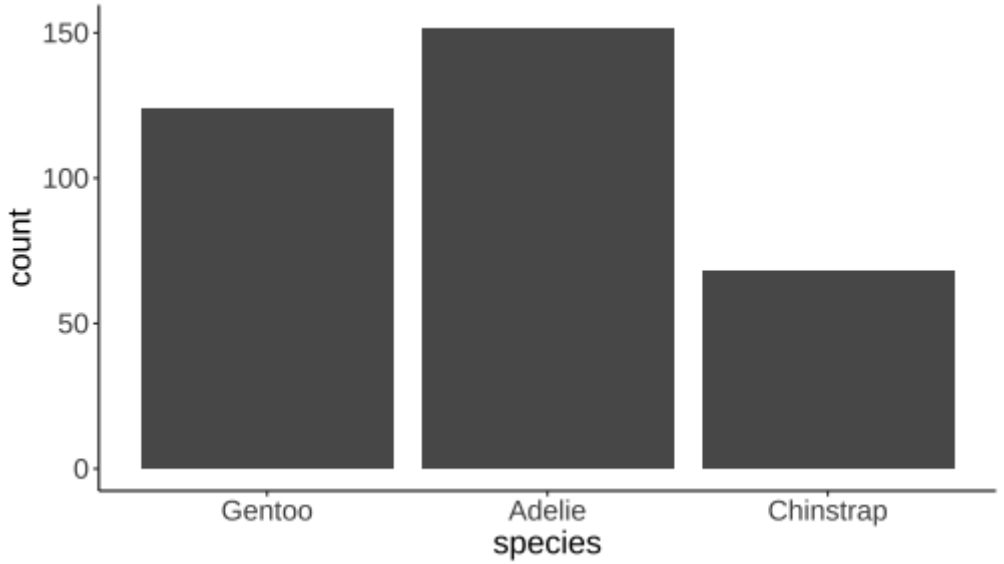
Providing only one level to `fct_relevel` makes that the base level (integer code 1).
You can also fully specify all the levels in order, or partially specify them. If you partially specify them, the remaining levels will be put in alphabetical order after the ones you specify.

Effect in plots

```
ggplot(penguins,
       aes(x = species))+
  geom_bar()
```




```
ggplot(p1,
       aes(x = species))+
  geom_bar()
```



Changes the order in which bars are plotted

Extra levels

```
x <- factor(str_split('statistics', '')[[1]],
            levels = letters)
x
```

```
 [1] s t a t i s t i c s
Levels: a b c d e f g h i j k l m n o p q r s t
```

```
p1 <- penguins %>% filter(species != 'Gentoo')
fct_count(p1$species)
```

```
# A tibble: 3 x 2
  f             n
  <fct>     <int>
1 Adelie      152
2 Chinstrap    68
3 Gentoo        0
```

```
fct_drop(x)
```

```
 [1] s t a t i s t i c s
Levels: a c i s t
```

```
p1 <- p1 %>% mutate(species = fct_drop(species
fct_count(p1$species)
```

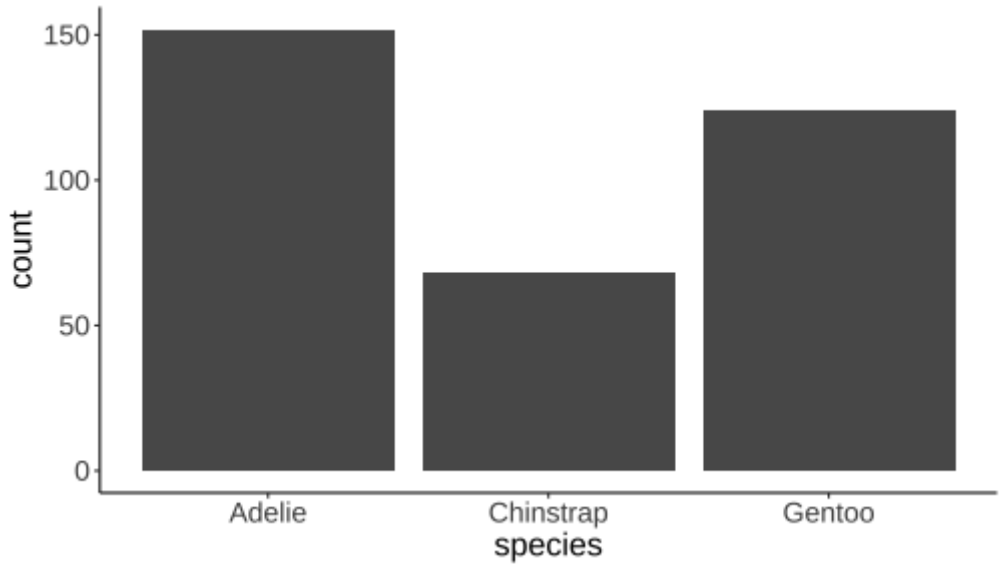
```
# A tibble: 2 x 2
  f             n
  <fct>     <int>
1 Adelie      152
2 Chinstrap    68
```

Getting rid of extra levels

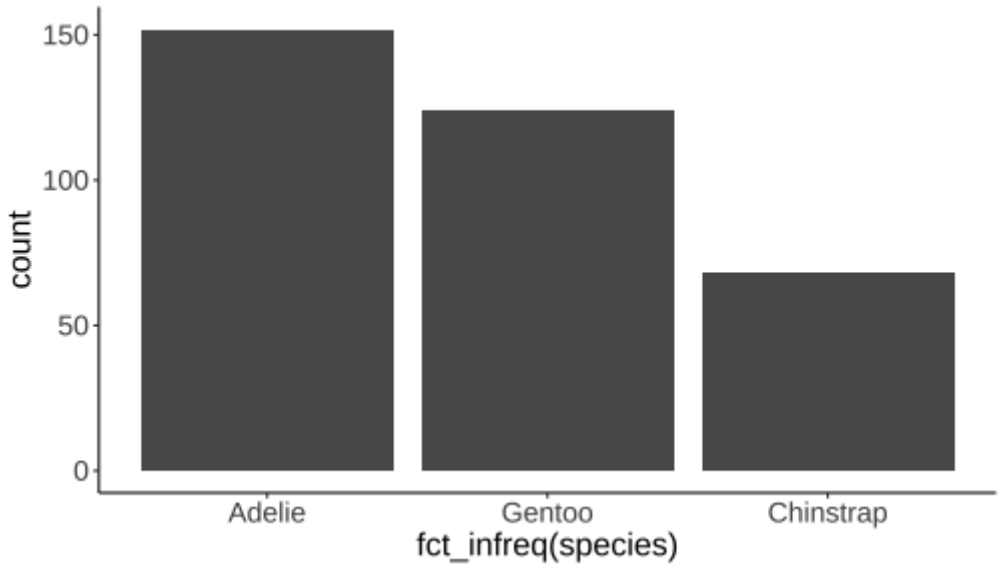
Sometimes levels with no data show up in summaries or plots

Ordering levels by frequency

```
ggplot(penguins,
       aes(x = species))+
  geom_bar()
```



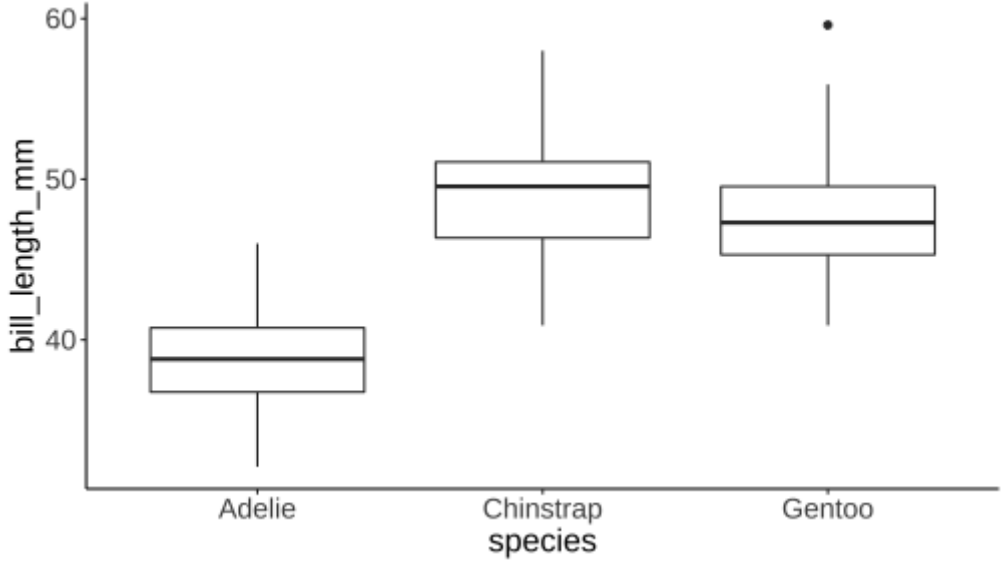

```
ggplot(penguins,
       aes(x = fct_infreq(species)))+
  geom_bar()
```



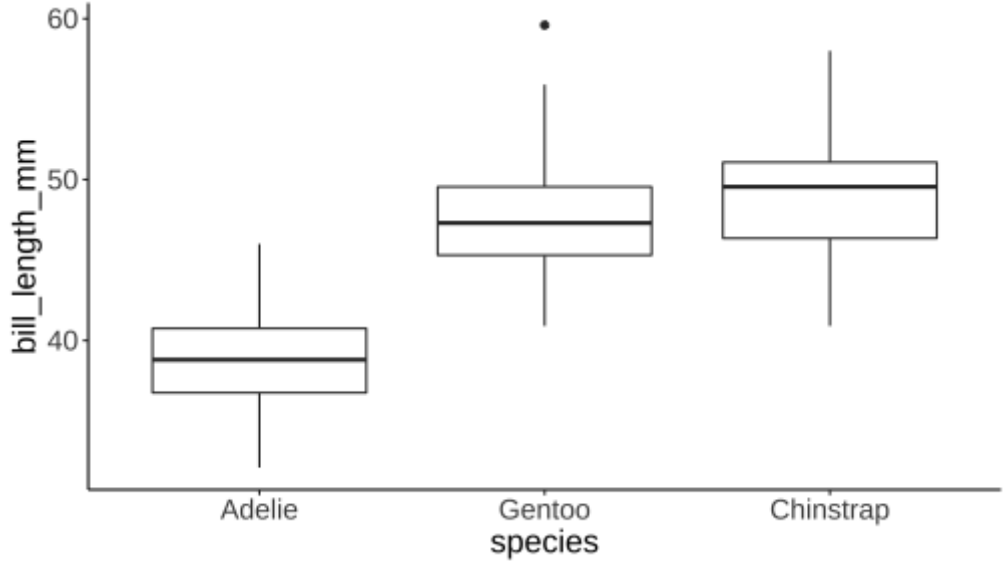
Ordering levels from most to least frequent

Ordering levels by values of another variable

```
ggplot(penguins,
       aes(x = species,
           y = bill_length_mm))+
  geom_boxplot()
```




```
ggplot(penguins,
       aes(x = fct_reorder(species, bill_length
                           .fun=median, na.rm=
           y = bill_length_mm))+
  geom_boxplot() + labs(x = 'species')
```



`fct_reorder` is useful for ordering a plot by ascending or descending levels. This makes the plot easier to read.

Ordering levels by values of another variable

```
USArrests <- USArrests %>% rownames_to_column('State')
```

```
ggplot(USArrests, aes(x=State, y = Murder))+
  geom_bar(stat = 'identity') +
  theme(axis.text = element_text(size=6))+
  coord_flip()
```

```
ggplot(USArrests, aes(
  x = fct_reorder(State, Murder),
  y = Murder))+
  geom_bar(stat = 'identity')+
  theme(axis.text = element_text(size=6))+
  coord_flip()
```

Order levels based on last values when plotting 2 variables

The level ordering also shows up in the order of levels in the legends of plots. Suppose you are plotting two variables, grouped by a factor.

```
ggplot(iris, aes(
  x = Sepal.Length,
  y = Sepal.Width,
  color = Species))+
  geom_smooth(se=F)
```

```
ggplot(iris, aes(
  x = Sepal.Length,
  y = Sepal.Width,
  color = fct_reorder2(Species,
                       Sepal.Length, Sepal.Wid
  geom_smooth(se=F) + labs(color = 'Species')
```

Further exploration

1. forcats cheatsheet
2. Chapter 15 of R4DS

For loops

For-loops are a computational structure that allows you to do the same thing repeatedly over a loop with some index.

The basic structure is

```
for (variable in vector) {
  <code to execute for each iteration>
}
```

Lists

Directly using lists has efficiency advantages. `rio` can load all the datasets into a list, for example.

```
sites <- c('Brain','Colon','Esophagus','Lung','Oral')
dats <- rio::import_list(file.path(paste0(sites,'.csv')))
names(dats)
```

```
[1] "Brain"     "Colon"     "Esophagus" "Lung"      "Oral"
```

```
str(dats[['Brain']])
```

```
'data.frame':    43 obs. of  10 variables:
 $ Year of Diagnosis   : chr  "1975-2016" "1975" "1976" "1977" ...
 $ All Races,Both Sexes: num  6.59 5.85 5.82 6.17 5.76 6.12 6.3 6.51 6.42 6.31 ...
 $ All Races,Males     : num  7.88 6.84 7.14 7.76 6.79 7.42 7.58 8.07 7.93 7.6 ...
 $ All Races,Females   : num  5.51 5.01 4.68 4.89 4.91 5.01 5.24 5.2 5.24 5.19 ...
 $ Whites,Both Sexes   : num  7.22 6.21 6.18 6.6 6.1 6.6 6.81 6.9 6.92 6.88 ...
 $ Whites,Males        : num  8.61 7.31 7.51 8.26 7.19 8.03 8.2 8.44 8.57 8.2 ...
 $ Whites,Females      : num  6.04 5.28 5.03 5.27 5.19 5.37 5.65 5.63 5.64 5.74 ...
 $ Blacks,Both Sexes   : num  4.08 4.14 3.32 3.55 3.86 3.69 3.14 5.02 3.71 2.75 ...
 $ Blacks,Males        : num  4.79 4.31 5.37 5.17 4.34 4.19 3.35 7.24 4.4 3.79 ...
 $ Blacks,Females      : chr  "3.51" "3.88" "-" "2.47" ...
```

A note on `rio::import` for reading CSV files

The function `rio::import` reads CSV files using `data.table::fread`, and then converts the resulting `data.table` object into a `data.frame` object.

`fread` is not only really fast, but also makes some great automatic choices.

- It looks for and tries to omit non-standard header rows (so we don't need `skip=4`)
- It automatically tries to figure out the right number of rows to import
- With the `check.names=TRUE` option, it fixes issues with column names to make them conformant with R

Using `rio::import` solves a lot of troublesome things in importing regular text files (CSV, TSV, etc), and is **recommended**

Lists

```
dats <- rio::import_list(file.path('../data', paste0(sites,'.csv')),
                         setclass = 'tbl',     # Output as tibble
                         check.names = TRUE)   # Check and fix names
str(dats[['Brain']])
```

```
tibble [43 × 10] (S3: tbl_df/tbl/data.frame)
 $ Year.of.Diagnosis   : chr [1:43] "1975-2016" "1975" "1976" "1977" ...
 $ All.Races.Both.Sexes: num [1:43] 6.59 5.85 5.82 6.17 5.76 6.12 6.3 6.51 6.42 6.31 ...
 $ All.Races.Males     : num [1:43] 7.88 6.84 7.14 7.76 6.79 7.42 7.58 8.07 7.93 7.6 ...
 $ All.Races.Females   : num [1:43] 5.51 5.01 4.68 4.89 4.91 5.01 5.24 5.2 5.24 5.19 ...
 $ Whites.Both.Sexes   : num [1:43] 7.22 6.21 6.18 6.6 6.1 6.6 6.81 6.9 6.92 6.88 ...
 $ Whites.Males        : num [1:43] 8.61 7.31 7.51 8.26 7.19 8.03 8.2 8.44 8.57 8.2 ...
 $ Whites.Females      : num [1:43] 6.04 5.28 5.03 5.27 5.19 5.37 5.65 5.63 5.64 5.74 ...
 $ Blacks.Both.Sexes   : num [1:43] 4.08 4.14 3.32 3.55 3.86 3.69 3.14 5.02 3.71 2.75 ...
 $ Blacks.Males        : num [1:43] 4.79 4.31 5.37 5.17 4.34 4.19 3.35 7.24 4.4 3.79 ...
 $ Blacks.Females      : chr [1:43] "3.51" "3.88" "-" "2.47" ...
```

purrr::map

Map

`map` is like a for-loop, but strictly for lists. It is more efficient than for-loops. The basic templates are:

```
map(<list>, <function>, <function arguments>)
map(<list>, <function>(<arguments>){<definiition>})
map(<list>, ~ <definition with .x placeholder>)
```

with the first argument of the function being the entry point for each component of the list (or replacing the `.x` placeholder)

For example, if we want to take out the first row of each dataset and make sure all the variables are numeric, we could do:

```
library(tidyverse)
dats <- map(dats,
            function(d){
              d %>% dplyr::slice(-1) %>%  # remove first row which contains summary data
                mutate(across(where(is.character), as.numeric))
            })
str(dats[['Brain']])
```

```
tibble [42 × 10] (S3: tbl_df/tbl/data.frame)
 $ Year.of.Diagnosis   : num [1:42] 1975 1976 1977 1978 1979 ...
 $ All.Races.Both.Sexes: num [1:42] 5.85 5.82 6.17 5.76 6.12 6.3 6.51 6.42 6.31 6.12 ...
```

I don't like the names with dots, say 😄. I can just apply a function to each data set to fix that.

```
dats <- map(dats, janitor::clean_names) # assumes first argument gets elements of dats
str(dats[['Oral']])
```

```
tibble [42 × 10] (S3: tbl_df/tbl/data.frame)
 $ year_of_diagnosis   : num [1:42] 1975 1976 1977 1978 1979 ...
 $ all_races_both_sexes: num [1:42] 13.2 13.3 12.7 13.4 14 ...
 $ all_races_males     : num [1:42] 21.2 21 20.1 20.9 21.9 ...
 $ all_races_females   : num [1:42] 7.09 7.39 6.94 7.71 7.98 7.91 7.91 7.93 7.24 7.86 ...
 $ whites_both_sexes   : num [1:42] 13.3 13.2 12.6 13.2 13.7 ...
 $ whites_males        : num [1:42] 21.7 21.1 19.9 20.7 21.6 ...
 $ whites_females      : num [1:42] 6.94 7.38 7 7.57 7.72 7.62 7.95 7.85 7.28 7.64 ...
 $ blacks_both_sexes   : num [1:42] 13.4 15.2 14.5 15.9 18.5 ...
 $ blacks_males        : num [1:42] 20.2 23.8 23.9 26 28.2 ...
 $ blacks_females      : num [1:42] 8.23 8.37 6.77 8.18 10.77 ...
```

Note that `janitor::clean_names` takes a data.frame/tibble as its first argument (as all tidyverse functions), and `dats` is a list of tibbles. So `map` applies the `clean_names` function to each tibble in the list, and returns the result as a list

Now let's split up by sexes

```
dats_all <- map(dats, select, year_of_diagnosis, ends_with('sexes'))
dats_male <- map(dats, select, year_of_diagnosis, ends_with('_males'))
dats_female <- map(dats, select, year_of_diagnosis, ends_with('females'))
str(dats_all[['Esophagus']])
```

```
tibble [42 × 4] (S3: tbl_df/tbl/data.frame)
 $ year_of_diagnosis   : num [1:42] 1975 1976 1977 1978 1979 ...
 $ all_races_both_sexes: num [1:42] 4.14 4.3 4.06 4.12 4.42 4.27 4.14 4.26 4.29 4.18 ...
 $ whites_both_sexes   : num [1:42] 3.55 3.72 3.33 3.41 3.73 3.54 3.31 3.46 3.57 3.52 ...
 $ blacks_both_sexes   : num [1:42] 10.9 10.7 12 13.1 12.9 ...
```

Here I used the form `map(<list>, <function>, <function arguments>)`.

Earlier I had used `map(<list>,<function definition>)` and `map(<list>, <function>)` with no (i.e., default) arguments.

Note, `map` assumes that each element of the list is the **first** argument of the function, and so you only have to specify from the 2nd argument onwards

Let's make the column headers of each dataset reflect the site, so that when we join we can keep the sites separate

```
for(n in sites){
  names(dats_all[[n]]) <- str_replace(names(dats_all[[n]]), 'both_sexes',n)
  names(dats_male[[n]]) <- str_replace(names(dats_male[[n]]), 'male',n)
  names(dats_female[[n]]) <- str_replace(names(dats_female[[n]]), 'female',n)
}
names(dats_all[['Esophagus']])
```

```
[1] "year_of_diagnosis"   "all_races_Esophagus" "whites_Esophagus"
[4] "blacks_Esophagus"
```

When we joined these data sets, we had to repeatedly use `left_join` to create the final data set.

```
joined_all <- dats_all[['Brain']]
for(n in setdiff(names(dats2_all), 'Brain')){
  joined_all <- joined %>% left_join(dats_all[
}
```

There is a shortcut to this repeated operation of a function with two inputs as applied to a list successively.

```
joined_all <- Reduce(left_join, dats_all)
joined_male <- Reduce(left_join, dats_male)
joined_female <- Reduce(left_join, dats_female
```

```
tibble [42 × 16] (S3: tbl_df/tbl/data.frame)
 $ year_of_diagnosis  : num [1:42] 1975 1976 1977 1978 1979 ...
 $ all_races_Brain    : num [1:42] 5.85 5.82 6.17 5.76 6.12 6.3 6.51 6.42 6.31 6.12 ...
 $ whites_Brain       : num [1:42] 6.21 6.18 6.6 6.1 6.6 6.81 6.9 6.92 6.88 6.49 ...
 $ blacks_Brain       : num [1:42] 4.14 3.32 3.55 3.86 3.69 3.14 5.02 3.71 2.75 4.53 ...
 $ all_races_Colon    : num [1:42] 59.5 61.3 62.4 62 62.4 ...
 $ whites_Colon       : num [1:42] 60.2 62.2 63.2 62.8 63 ...
 $ blacks_Colon       : num [1:42] 56.9 55 60.8 62.2 58.6 ...
 $ all_races_Esophagus: num [1:42] 4.14 4.3 4.06 4.12 4.42 4.27 4.14 4.26 4.29 4.18 ...
 $ whites_Esophagus   : num [1:42] 3.55 3.72 3.33 3.41 3.73 3.54 3.31 3.46 3.57 3.52 ...
 $ blacks_Esophagus   : num [1:42] 10.9 10.7 12 13.1 12.9 ...
 $ all_races_Lung     : num [1:42] 52.2 55.4 56.7 57.8 58.6 ...
 $ whites_Lung        : num [1:42] 51.9 54.6 55.9 57.2 58 ...
 $ blacks_Lung        : num [1:42] 64.5 72.3 73.6 74.4 74.5 ...
 $ all_races_Oral     : num [1:42] 13.2 13.3 12.7 13.4 14 ...
 $ whites_Oral        : num [1:42] 13.3 13.2 12.6 13.2 13.7 ...
 $ blacks_Oral        : num [1:42] 13.4 15.2 14.5 15.9 18.5 ...
```

Next, we want to separate the races from the sites, after a `pivot_longer`. The `all_races` will pose a problem if we split on `_`. Let's fix that.

```
names(joined_all) <- str_replace(names(joined_all), 'all_races','allraces')
names(joined_male) <- str_replace(names(joined_male), 'all_races','allraces')
names(joined_female) <- str_replace(names(joined_female), 'all_races','allraces')
```

Now, for each of these, we need to gather then separate. We'll put the data sets in a list first

```
joined <- list('both'=joined_all, 'male'=joined_male, 'female'=joined_female)
joined <- map(joined,
              function(d){
                d %>%
                 pivot_longer(names_to='variable', values_to = 'rate',
                              cols=c(-year_of_diagnosis)) %>%
                  separate(variable, c('race','site'), sep='_')
              })
str(joined[['both']])
```

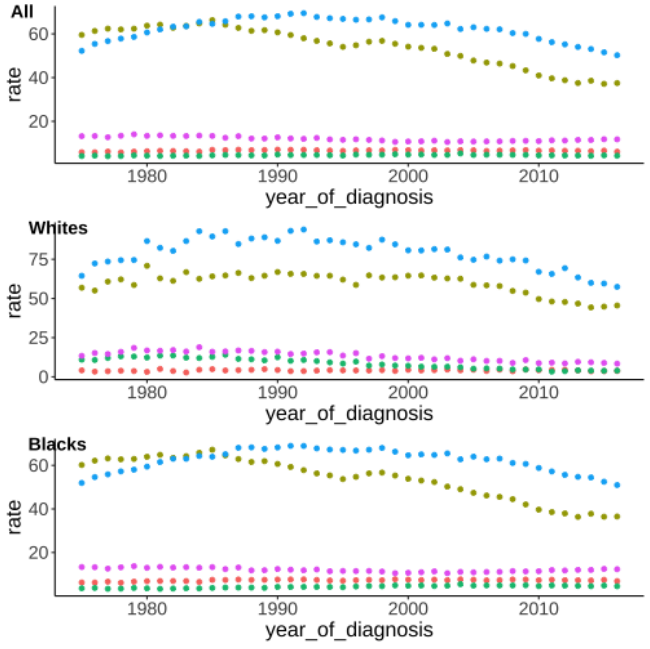
```
tibble [630 × 4] (S3: tbl_df/tbl/data.frame)
 $ year_of_diagnosis: num [1:630] 1975 1975 1975 1975 1975 ...
 $ race             : chr [1:630] "allraces" "whites" "blacks" "allraces" ...
 $ site             : chr [1:630] "Brain" "Brain" "Brain" "Colon" ...
 $ rate             : num [1:630] 5.85 6.21 4.14 59.54 60.2 ...
```

Okay, this is voodoo 😈 Not really. Grab one of the datasets and work out what you need. Since you'll be doing the same to all of the datasets, you use `map` on the list of datasets

Final graphing

Now we're in a position to do the graphing.

```
pltlist <- joined[['both']] %>% group_split(race) %>%
  map(function(d) {ggplot(d,
                          aes(x = year_of_diagnosis,
                              y = rate,
                              color=site))+
  geom_point(show.legend = F) })
cowplot::plot_grid(plotlist=pltlist, ncol=1,
                   labels=c('All','Whites','Blacks'))
```



I'm using quite advanced R here, but hopefully you'll learn by example.

`group_split` splits the dataset by the values of the grouping variable into a list

Statistical summaries

Where we're going

1. Creating data summaries
2. Basic statistical comparisons between groups
3. Creating tables
    - Table 1
    - Tables for analytic results

The basic assumption we'll make is that we will start with a tidy data set.

Statistical summaries

Univariate summaries

Single summaries

- Mean (`mean`)
- Variance(`var`)
- Standard deviation (`sd`)
- Count (`nrow` or `dplyr::n` or `dplyr::n_distinct`)
- Median (`median`)
- Inter-quartile range (`IQR`)
- Mean absolute deviation (`mad`)
- Minimum (`min`) and Maximum (`max`)

Multiple summaries

- Quantiles (`quantile`)
- Range (`range`)

Summarizing the breast cancer expression dataset

Mean

```
brca <- rio::import('../../data/BreastCancer_Expression.csv')
brca %>%
  summarize(across(starts_with('NP'),
                   mean, na.rm=T))
```

```
  NP_958782 NP_958785 NP_958786 NP_000436 NP_958781 NP_958780 NP_958783
1 0.3202321 0.3269153 0.3264254 0.3236833 0.3270832 0.3263382 0.3259212
  NP_958784  NP_112598 NP_001611
1 0.3259995 -0.3074577 0.4578748
```

Median

```
brca %>%
  summarize(across(starts_with('NP'),
                   median, na.rm=T))
```

```
  NP_958782 NP_958785 NP_958786 NP_000436 NP_958781 NP_958780 NP_958783
1 0.3236627 0.3269726 0.3269726 0.3302826 0.3269726 0.3269726 0.3269726
  NP_958784  NP_112598 NP_001611
1 0.3269726 -0.6021319 0.6948104
```

Standard deviation

```
brca %>%
  summarize(across(starts_with('NP'),
                   sd, na.rm=T))
```

```
  NP_958782 NP_958785 NP_958786 NP_000436 NP_958781 NP_958780 NP_958783
1 0.9767777 0.9800721 0.9799358 0.9784656 0.9806001 0.9796277 0.9806739
  NP_958784 NP_112598 NP_001611
1 0.9807512  2.024663  1.496951
```

Multiple summaries together

```
brca %>%
  summarize(across(starts_with('NP'),
                   c(mean,
                     median,
                     sd), na.rm=T))
```

```
  NP_958782_1 NP_958782_2 NP_958782_3 NP_958785_1 NP_958785_2 NP_958785_3
1   0.3202321   0.3236627   0.9767777   0.3269153   0.3269726   0.9800721
  NP_958786_1 NP_958786_2 NP_958786_3 NP_000436_1 NP_000436_2 NP_000436_3
1   0.3264254   0.3269726   0.9799358   0.3236833   0.3302826   0.9784656
  NP_958781_1 NP_958781_2 NP_958781_3 NP_958780_1 NP_958780_2 NP_958780_3
1   0.3270832   0.3269726   0.9806001   0.3263382   0.3269726   0.9796277
  NP_958783_1 NP_958783_2 NP_958783_3 NP_958784_1 NP_958784_2 NP_958784_3
1   0.3259212   0.3269726   0.9806739   0.3259995   0.3269726   0.9807512
  NP_112598_1 NP_112598_2 NP_112598_3 NP_001611_1 NP_001611_2 NP_001611_3
1  -0.3074577  -0.6021319    2.024663   0.4578748   0.6948104    1.496951
```

Multiple summaries together

```
brca %>%
  summarize(across(-1, # got tired of typing
                   c('Mean'=mean,
                     'Median' = median,
                     'SD'=sd), na.rm=T))
```

```
  NP_958782_Mean NP_958782_Median NP_958782_SD NP_958785_Mean NP_958785_Median
1      0.3202321        0.3236627    0.9767777      0.3269153        0.3269726
  NP_958785_SD NP_958786_Mean NP_958786_Median NP_958786_SD NP_000436_Mean
1    0.9800721      0.3264254        0.3269726    0.9799358      0.3236833
  NP_000436_Median NP_000436_SD NP_958781_Mean NP_958781_Median NP_958781_SD
1        0.3302826    0.9784656      0.3270832        0.3269726    0.9806001
  NP_958780_Mean NP_958780_Median NP_958780_SD NP_958783_Mean NP_958783_Median
1      0.3263382        0.3269726    0.9796277      0.3259212        0.3269726
  NP_958783_SD NP_958784_Mean NP_958784_Median NP_958784_SD NP_112598_Mean
1    0.9806739      0.3259995        0.3269726    0.9807512     -0.3074577
  NP_112598_Median NP_112598_SD NP_001611_Mean NP_001611_Median NP_001611_SD
1       -0.6021319     2.024663      0.4578748        0.6948104     1.496951
```

Multiple summaries together

```
brca %>%
  summarize(across(-1,
                   c('Mean' = mean,
                     'Median' = median,
                     'SD' = sd), na.rm=T)) %>%
  pivot_longer(cols=everything(),
                   names_to='variable',
                   values_to='value') %>%
  # extract(variable, c('ID','Statistic'),
          # regex = '(NP_\\d+)_([A-Za-z]+)') %>%
  separate(variable,
           c("Type",'ID','Statistic'), sep='_') %>%
  pivot_wider(names_from = Statistic, values_from = value) %>%
  unite(ID, c('Type','ID'), sep='_')
```

```
# A tibble: 10 x 4
   ID          Mean Median    SD
   <chr>      <dbl>  <dbl> <dbl>
 1 NP_958782  0.320  0.324 0.977
 2 NP_958785  0.327  0.327 0.980
 3 NP_958786  0.326  0.327 0.980
 4 NP_000436  0.324  0.330 0.978
 5 NP_958781  0.327  0.327 0.981
 6 NP_958780  0.326  0.327 0.980
```

You could replace the highlighted code with

```
extract(variable,
        c('ID','Statistic'),
        regex = '(NP_\\d+)_([
  pivot_wider(
    names_from=Statistic,
    values_from=value)
```

Summarizing a data set

Data set summary

There is a function `summary` that will give you summaries of all the variables. It's nice for looking at the data, but the output format isn't very good for further manipulation

```
summary(brca[,-1]) # Omit first column
```

```
   NP_958782          NP_958785          NP_958786          NP_000436
 Min.   :-1.9478   Min.   :-1.9527   Min.   :-1.9552   Min.   :-1.9478
 1st Qu.:-0.4549   1st Qu.:-0.4421   1st Qu.:-0.4440   1st Qu.:-0.4385
 Median : 0.3237   Median : 0.3270   Median : 0.3270   Median : 0.3303
 Mean   : 0.3202   Mean   : 0.3269   Mean   : 0.3264   Mean   : 0.3237
 3rd Qu.: 0.9181   3rd Qu.: 0.9238   3rd Qu.: 0.9238   3rd Qu.: 0.9180
 Max.   : 2.7651   Max.   : 2.7797   Max.   : 2.7797   Max.   : 2.7980
   NP_958781          NP_958780          NP_958783          NP_958784
 Min.   :-1.9576   Min.   :-1.9552   Min.   :-1.9552   Min.   :-1.9552
 1st Qu.:-0.4440   1st Qu.:-0.4458   1st Qu.:-0.4440   1st Qu.:-0.4440
 Median : 0.3270   Median : 0.3270   Median : 0.3270   Median : 0.3270
 Mean   : 0.3271   Mean   : 0.3263   Mean   : 0.3259   Mean   : 0.3260
 3rd Qu.: 0.9277   3rd Qu.: 0.9238   3rd Qu.: 0.9238   3rd Qu.: 0.9238
 Max.   : 2.7870   Max.   : 2.7797   Max.   : 2.7834   Max.   : 2.7834
   NP_112598          NP_001611
 Min.   :-4.9527   Min.   :-2.5751
 1st Qu.:-1.6741   1st Qu.:-0.5216
 Median :-0.6021   Median : 0.6948
 Mean   :-0.3075   Mean   : 0.4579
 3rd Qu.: 0.8696   3rd Qu.: 1.4394
```

Maybe an easier way?

The `tableone` package

The `tableone` package is meant to create, you guessed it, Table 1.

It is quite a convenient package for most purposes and saves gobs of time

The tableone package

```
library(tableone)
tab1 <- CreateTableOne(data=brca[,-1])
tab1
```

```
                          Overall
  n                          83
  NP_958782 (mean (SD))   0.32 (0.98)
  NP_958785 (mean (SD))   0.33 (0.98)
  NP_958786 (mean (SD))   0.33 (0.98)
  NP_000436 (mean (SD))   0.32 (0.98)
  NP_958781 (mean (SD))   0.33 (0.98)
  NP_958780 (mean (SD))   0.33 (0.98)
  NP_958783 (mean (SD))   0.33 (0.98)
  NP_958784 (mean (SD))   0.33 (0.98)
  NP_112598 (mean (SD)) -0.31 (2.02)
  NP_001611 (mean (SD))   0.46 (1.50)
```

The tableone package

```
library(tableone)
tab1 <- CreateTableOne(data = brca[-1])
print(tab1, nonnormal = names(brca)[-1])
```

You have to give the variable names of those you think are non-normally distributed and need to be summarized by the median

```
                              Overall
  n                             83
  NP_958782 (median [IQR])     0.32 [-0.45, 0.92]
  NP_958785 (median [IQR])     0.33 [-0.44, 0.92]
  NP_958786 (median [IQR])     0.33 [-0.44, 0.92]
  NP_000436 (median [IQR])     0.33 [-0.44, 0.92]
  NP_958781 (median [IQR])     0.33 [-0.44, 0.93]
  NP_958780 (median [IQR])     0.33 [-0.45, 0.92]
  NP_958783 (median [IQR])     0.33 [-0.44, 0.92]
  NP_958784 (median [IQR])     0.33 [-0.44, 0.92]
  NP_112598 (median [IQR])    -0.60 [-1.67, 0.87]
  NP_001611 (median [IQR])     0.69 [-0.52, 1.44]
```

The tableone package

```
library(tableone)
tab1 <- CreateTableOne(data = brca[-1])
kableone(print(tab1, nonnormal = names(brca)[-1
         format='html')
```

	Overall
n	83
NP_958782 (median [IQR])	0.32 [-0.45, 0.92]
NP_958785 (median [IQR])	0.33 [-0.44, 0.92]
NP_958786 (median [IQR])	0.33 [-0.44, 0.92]
NP_000436 (median [IQR])	0.33 [-0.44, 0.92]
NP_958781 (median [IQR])	0.33 [-0.44, 0.93]
NP_958780 (median [IQR])	0.33 [-0.45, 0.92]
NP_958783 (median [IQR])	0.33 [-0.44, 0.92]
NP_958784 (median [IQR])	0.33 [-0.44, 0.92]
NP_112598 (median [IQR])	-0.60 [-1.67, 0.87]
NP_001611 (median [IQR])	0.69 [-0.52, 1.44]

Mixed data

Let's first put the expression and clinical data together

```
library(rio)
brca1 <- import('../../data/clinical_data_breast_cancer_hw.csv')
brca2 <- import('../../data/BreastCancer_Expression.csv')
brca <- left_join(brca1, brca2, by=c('Complete.TCGA.ID' = 'TCGA_ID')) %>%
  mutate(Age.at.Initial.Pathologic.Diagnosis =
           as.numeric(Age.at.Initial.Pathologic.Diagnosis)) %>%
  mutate(ER.Status = ifelse(ER.Status %in% c('Positive','Negative'),
                            ER.Status, NA))

summary(brca)
```

```
 Complete.TCGA.ID     Gender          Age.at.Initial.Pathologic.Diagnosis
 Length:108         Length:108         Min.   :30.00
 Class :character   Class :character   1st Qu.:49.00
 Mode  :character   Mode  :character   Median :58.00
                                       Mean   :58.72
                                       3rd Qu.:66.50
                                       Max.   :88.00
                                       NA's   :1
  ER.Status          PR.Status          HER2.Final.Status     Tumor
 Length:108         Length:108         Length:108         Length:108
 Class :character   Class :character   Class :character   Class :character
 Mode  :character   Mode  :character   Mode  :character   Mode  :character



    Node             Metastasis         AJCC.Stage         Vital.Status
 Length:108         Length:108         Length:108         Length:108
```

Let's first put the expression and clinical data together

```
library(rio)
brca1 <- import('../../data/clinical_data_breast_cancer_hw.csv')
brca2 <- import('../../data/BreastCancer_Expression.csv')
brca <- left_join(brca1, brca2, by=c('Complete.TCGA.ID' = 'TCGA_ID')) %>%
  mutate(Age.at.Initial.Pathologic.Diagnosis =
           as.numeric(Age.at.Initial.Pathologic.Diagnosis)) %>%
  mutate(ER.Status = ifelse(ER.Status %in% c('Positive','Negative'),
                            ER.Status, NA),
         HER2.Final.Status = ifelse(HER2.Final.Status=='Equivocal',
                                    NA, HER2.Final.Status)) %>%
  mutate(across(is.character, as.factor)) %>%
  mutate(Complete.TCGA.ID = as.character(Complete.TCGA.ID))

str(brca)
```

```
'data.frame':    108 obs. of  23 variables:
 $ Complete.TCGA.ID                : chr  "TCGA-A2-A0T2" "TCGA-A2-A0CM" "TCGA-BH-A18V" "TCGA-BH-A
 $ Gender                          : Factor w/ 2 levels "FEMALE","MALE": 1 1 1 1 1 1 1 1 1 1 ...
 $ Age.at.Initial.Pathologic.Diagnosis: num  66 40 48 56 38 57 74 60 61 NA ...
 $ ER.Status                       : Factor w/ 2 levels "Negative","Positive": 1 1 1 1 1 1 1 1 1
 $ PR.Status                       : Factor w/ 2 levels "Negative","Positive": 1 1 1 1 1 1 1 1 1
 $ HER2.Final.Status               : Factor w/ 2 levels "Negative","Positive": 1 1 1 1 1 1 1 1 1
 $ Tumor                           : Factor w/ 4 levels "T1","T2","T3",..: 3 2 2 2 3 2 3 2 2 2 .
 $ Node                            : Factor w/ 4 levels "N0","N1","N2",..: 4 1 2 2 4 1 1 1 1 1 .
 $ Metastasis                      : Factor w/ 2 levels "M0","M1": 2 1 1 1 1 1 1 1 1 1 ...
 $ AJCC.Stage                      : Factor w/ 11 levels "Stage I","Stage IA",..: 11 5 6 6 10 5 6
 $ Vital.Status                    : Factor w/ 2 levels "DECEASED","LIVING": 1 1 1 1 2 2 2 2 2 2
 $ Days.to.Date.of.Last.Contact    : int  240 754 1555 1692 133 309 425 643 775 964 ...
```

Identify which variables are categorical (factors) and which are continuous (numeric)

```
catvars <- brca %>% select(where(is.factor)) %>% names()
ctsvars <- brca %>% select(where(is.numeric)) %>% names()
```

```
CreateCatTable(vars = catvars, data = brca)
```

```
                                  Overall
  n                               108
  Gender = MALE (%)                2 ( 1.9)
  ER.Status = Positive (%)        69 (64.5)
  PR.Status = Positive (%)        55 (50.9)
  HER2.Final.Status = Positive (%) 28 (26.2)
  Tumor (%)
     T1                           16 (14.8)
     T2                           67 (62.0)
     T3                           19 (17.6)
     T4                            6 ( 5.6)
  Node (%)
     N0                           54 (50.0)
     N1                           30 (27.8)
     N2                           15 (13.9)
     N3                            9 ( 8.3)
  Metastasis = M1 (%)              2 ( 1.9)
  AJCC.Stage (%)
     Stage I                       3 ( 2.8)
     Stage IA                      7 ( 6.5)
     Stage IB                      2 ( 1.9)
     Stage II                     11 (10.2)
     Stage IIA                    32 (29.6)
     Stage IIB                    23 (21.3)
     Stage III                     4 ( 3.7)
     Stage IIIA                   12 (11.1)
     Stage IIIB                    6 ( 5.6)
```

```
CreateContTable(vars = ctsvars, data = brca)
```

```
                                                Overall
  n                                             108
  Age.at.Initial.Pathologic.Diagnosis (mean (SD))  58.72 (13.21)
  Days.to.Date.of.Last.Contact (mean (SD))        806.37 (667.70)
  Days.to.date.of.Death (mean (SD))              1254.45 (678.05)
  NP_958782 (mean (SD))                             0.32 (0.99)
  NP_958785 (mean (SD))                             0.33 (1.00)
  NP_958786 (mean (SD))                             0.33 (1.00)
  NP_000436 (mean (SD))                             0.32 (0.99)
  NP_958781 (mean (SD))                             0.33 (1.00)
  NP_958780 (mean (SD))                             0.33 (1.00)
  NP_958783 (mean (SD))                             0.33 (1.00)
  NP_958784 (mean (SD))                             0.33 (1.00)
  NP_112598 (mean (SD))                            -0.30 (2.06)
  NP_001611 (mean (SD))                             0.38 (1.46)
```

```r
brca <- brca %>%
  rename(
    'Age'='Age.at.Initial.Pathologic.Diagnosis
    'Last.Contact' = 'Days.to.Date.of.Last.Cont
    'Death' = 'Days.to.date.of.Death'
  )
ctsvars <- brca %>%
  select(where(is.numeric))%>% names()
CreateContTable(vars = ctsvars, data = brca)
```

```
                            Overall
  n                         108
  Age (mean (SD))             58.72 (13.21)
  Last.Contact (mean (SD))   806.37 (667.70)
  Death (mean (SD))         1254.45 (678.05)
  NP_958782 (mean (SD))        0.32 (0.99)
  NP_958785 (mean (SD))        0.33 (1.00)
  NP_958786 (mean (SD))        0.33 (1.00)
  NP_000436 (mean (SD))        0.32 (0.99)
  NP_958781 (mean (SD))        0.33 (1.00)
  NP_958780 (mean (SD))        0.33 (1.00)
  NP_958783 (mean (SD))        0.33 (1.00)
  NP_958784 (mean (SD))        0.33 (1.00)
  NP_112598 (mean (SD))       -0.30 (2.06)
  NP_001611 (mean (SD))        0.38 (1.46)
```

Putting it together

```
CreateTableOne(vars = c(catvars, ctsvars),
               data = brca)
```

```
                                    Overall
  n                                   108
  Gender = MALE (%)                     2 ( 1.9)
  ER.Status = Positive (%)             69 (64.5)
  PR.Status = Positive (%)             55 (50.9)
  HER2.Final.Status = Positive (%)     28 (26.2)
  Tumor (%)
     T1                                16 (14.8)
     T2                                67 (62.0)
     T3                                19 (17.6)
     T4                                 6 ( 5.6)
  Node (%)
     N0                                54 (50.0)
     N1                                30 (27.8)
     N2                                15 (13.9)
     N3                                 9 ( 8.3)
  Metastasis = M1 (%)                   2 ( 1.9)
  AJCC.Stage (%)
     Stage I                            3 ( 2.8)
     Stage IA                           7 ( 6.5)
     Stage IB                           2 ( 1.9)
```

Putting it together

```
CreateTableOne(data = brca[,-1])
```

```
                                  Overall
  n                                 108
  Gender = MALE (%)                   2 ( 1.9)
  Age (mean (SD))                 58.72 (13.21)
  ER.Status = Positive (%)            69 (64.5)
  PR.Status = Positive (%)            55 (50.9)
  HER2.Final.Status = Positive (%)    28 (26.2)
  Tumor (%)
     T1                               16 (14.8)
     T2                               67 (62.0)
     T3                               19 (17.6)
     T4                                6 ( 5.6)
  Node (%)
     N0                               54 (50.0)
     N1                               30 (27.8)
     N2                               15 (13.9)
     N3                                9 ( 8.3)
  Metastasis = M1 (%)                  2 ( 1.9)
  AJCC.Stage (%)
     Stage I                           3 ( 2.8)
     Stage IA                          7 ( 6.5)
     Stage IB                          2 ( 1.9)
```

Grouped summaries

```
brca %>%
  group_by(ER.Status) %>%
  summarize(across(starts_with('NP'),
                   mean))
```

```
# A tibble: 3 x 11
  ER.Status NP_958782 NP_958785 NP_958786 NP_000436 NP_958781 NP_9
  <fct>          <dbl>     <dbl>     <dbl>     <dbl>     <dbl>
1 Negative          NA        NA        NA        NA        NA
2 Positive          NA        NA        NA        NA        NA
3 <NA>              NA        NA        NA        NA        NA
  NP_958783 NP_958784 NP_112598 NP_001611
      <dbl>     <dbl>     <dbl>     <dbl>
1        NA        NA        NA        NA
2        NA        NA        NA        NA
3        NA        NA        NA        NA
```

There are missing values now, so we have to use `na.rm=T`.

```
brca %>%
  group_by(ER.Status) %>%
  summarize(across(starts_with('NP'),
                   mean, na.rm=T))
```

```
# A tibble: 3 x 11
  ER.Status NP_958782 NP_958785 NP_958786 NP_000436 NP_958781 NP_95
  <fct>          <dbl>     <dbl>     <dbl>     <dbl>     <dbl>
1 Negative       0.429     0.438     0.439     0.432     0.436
2 Positive       0.267     0.273     0.272     0.271     0.274
3 <NA>         NaN       NaN       NaN       NaN       NaN       NaN
  NP_958783 NP_958784 NP_112598 NP_001611
      <dbl>     <dbl>     <dbl>     <dbl>
1     0.436     0.436    -0.197    -0.566
2     0.272     0.273    -0.357     0.840
3   NaN       NaN       NaN       NaN
```

We still have a row for the missing values of ER.Status

```
brca %>%
  filter(!is.na(ER.Status)) %>%
  group_by(ER.Status) %>%
  summarize(across(starts_with('NP'),
                   mean, na.rm=T))
```

```
# A tibble: 2 x 11
  ER.Status NP_958782 NP_958785 NP_958786 NP_000436 NP_958781 NP_9
  <fct>          <dbl>     <dbl>     <dbl>     <dbl>     <dbl>
1 Negative       0.429     0.438     0.439     0.432     0.436
2 Positive       0.267     0.273     0.272     0.271     0.274
  NP_958783 NP_958784 NP_112598 NP_001611
      <dbl>     <dbl>     <dbl>     <dbl>
1     0.436     0.436    -0.197    -0.566
2     0.272     0.273    -0.357     0.840
```

How about reversing the rows and columns for readability

```r
brca %>%
  filter(!is.na(ER.Status)) %>%
  group_by(ER.Status) %>%
  summarize(across(starts_with('NP'),
                   mean, na.rm=T)) %>%
  pivot_longer(names_to='ID', values_to='value
                cols = c(-ER.Status)) %>%
  pivot_wider(names_from = ER.Status,
              values_from=value)
```

```
# A tibble: 10 x 3
   ID        Negative Positive
   <chr>        <dbl>    <dbl>
 1 NP_958782    0.429    0.267
 2 NP_958785    0.438    0.273
 3 NP_958786    0.439    0.272
 4 NP_000436    0.432    0.271
 5 NP_958781    0.436    0.274
 6 NP_958780    0.436    0.273
 7 NP_958783    0.436    0.272
 8 NP_958784    0.436    0.273
 9 NP_112598   -0.197   -0.357
10 NP_001611   -0.566    0.840
```

Using tableone

```
CreateTableOne(
  data = brca %>% filter(!is.na(ER.Status)),
  vars = brca %>%
    select(starts_with('NP')) %>%
    names(),
  strata = 'ER.Status', # single quotes, not backticks
  test = F)
```

```
                     Stratified by ER.Status
                      Negative     Positive
  n                      38           69
  NP_958782 (mean (SD))  0.43 (1.13)  0.27 (0.93)
  NP_958785 (mean (SD))  0.44 (1.14)  0.27 (0.93)
  NP_958786 (mean (SD))  0.44 (1.14)  0.27 (0.93)
  NP_000436 (mean (SD))  0.43 (1.14)  0.27 (0.93)
  NP_958781 (mean (SD))  0.44 (1.14)  0.27 (0.93)
  NP_958780 (mean (SD))  0.44 (1.14)  0.27 (0.93)
  NP_958783 (mean (SD))  0.44 (1.14)  0.27 (0.93)
  NP_958784 (mean (SD))  0.44 (1.14)  0.27 (0.93)
  NP_112598 (mean (SD)) -0.20 (2.28) -0.36 (1.97)
  NP_001611 (mean (SD)) -0.57 (1.54)  0.84 (1.19)
```

Alternatives to tableone

- table1
- gtsummary
- flextable
- arsenal

arsenal

```
library(arsenal)
summary(tableby(ER.Status ~ ., data = brca[,-1])) # Here . implies all other variables.
```

	Negative (N=38)	Positive (N=69)	Total (N=107)	p value
Gender				0.289
FEMALE	38 (100.0%)	67 (97.1%)	105 (98.1%)	
MALE	0 (0.0%)	2 (2.9%)	2 (1.9%)	
Age				0.101
N-Miss	1	0	1	
Mean (SD)	55.919 (12.269)	60.348 (13.573)	58.802 (13.245)	
Range	36.000 - 82.000	30.000 - 88.000	30.000 - 88.000	
PR.Status				< 0.001
Negative	38 (100.0%)	14 (20.3%)	52 (48.6%)	
Positive	0 (0.0%)	55 (79.7%)	55 (51.4%)	
HER2.Final.Status				0.281
N-Miss	0	1	1	

Hypothesis tests

Comparing two groups

The t-test

The t-test compares whether the mean of a variable differs between two groups.

It does assume the normal distribution for the data, but is robust to deviations from normality

Do not test for normality before doing the t-test. It isn't necessary and screws up your error rates

The t-test

In R, there is a convenient function `t.test`

```
t.test(NP_958782 ~ ER.Status, data = brca)
```

```
    Welch Two Sample t-test

data:  NP_958782 by ER.Status
t = 0.63522, df = 41.807, p-value = 0.5287
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.3523151  0.6759226
sample estimates:
mean in group Negative mean in group Positive
            0.4292798             0.2674761
```

Read the code as

"Do a t-test to see if (the mean of) `NP_958782` differs by `ER.Status`, where both are taken from the data set `brca`"

You can read the `~` as "by", as in "t-test of NP_958782 by ER.Status"

Using broom

The fact that `broom::tidy` makes the results of tests into tibbles is in fact extremely useful in high-throughput work

```
brca %>%
  select(ER.Status, starts_with('NP')) %>%
  pivot_longer(names_to = 'protein',
               values_to = 'expression',
               cols = c(-ER.Status)) %>%
  split(.$protein) %>%
  map(~broom::tidy(t.test(expression ~ ER.Status,
                          data=.))) %>%
  bind_rows(.id = 'Protein') %>%
  select(Protein, estimate, p.value, conf.low, conf.high)
```

```
# A tibble: 10 x 5
   Protein   estimate  p.value conf.lo
   <chr>        <dbl>    <dbl>    <dbl
 1 NP_000436    0.161 0.534      -0.35
 2 NP_001611   -1.41  0.000199   -2.10
 3 NP_112598    0.160 0.761      -0.89
 4 NP_958780    0.163 0.528      -0.35
 5 NP_958781    0.162 0.530      -0.35
 6 NP_958782    0.162 0.529      -0.35
 7 NP_958783    0.164 0.527      -0.35
 8 NP_958784    0.164 0.527      -0.35
 9 NP_958785    0.165 0.524      -0.35
10 NP_958786    0.166 0.520      -0.35
```

Back to testing

Wilcoxon test, nonparametric t-test

```
wilcox.test(NP_958782 ~ ER.Status, data=brca) %>%
  broom::tidy()
```

```
# A tibble: 1 x 4
  statistic p.value method
      <dbl>   <dbl> <chr>
1       755   0.590 Wilcoxon rank sum test with continuity correction
  alternative
  <chr>
1 two.sided
```

```

    Wilcoxon rank sum test with continuity correction

data:  NP_958782 by ER.Status
W = 755, p-value = 0.5897
alternative hypothesis: true location shift is not equal to 0
```

Wilcoxon test

```
brca %>%
  select(ER.Status, starts_with('NP')) %>%
  tidyr::gather(protein,expression, -ER.Status
  split(.$protein) %>%
  map(~broom::tidy(wilcox.test(expression ~ ER
                               data=.))) %>%
  bind_rows(.id='Protein') %>%
  select(Protein, p.value)
```

```
# A tibble: 10 x 2
   Protein     p.value
   <chr>         <dbl>
 1 NP_000436 0.583
 2 NP_001611 0.0000928
 3 NP_112598 0.939
 4 NP_958780 0.583
 5 NP_958781 0.576
 6 NP_958782 0.590
 7 NP_958783 0.583
 8 NP_958784 0.576
 9 NP_958785 0.576
10 NP_958786 0.576
```

Using tableone

```
CreateTableOne(
  data = brca %>% filter(!is.na(ER.Status)),
  vars = brca %>%
    select(starts_with('NP')) %>%
    names(),
  strata = 'ER.Status',
  test = T,
  testNormal = t.test
)
```

```
                     Stratified by ER.Status
                      Negative     Positive     p      test
  n                      38           69
  NP_958782 (mean (SD))  0.43 (1.13)  0.27 (0.93)  0.498
  NP_958785 (mean (SD))  0.44 (1.14)  0.27 (0.93)  0.492
  NP_958786 (mean (SD))  0.44 (1.14)  0.27 (0.93)  0.487
  NP_000436 (mean (SD))  0.43 (1.14)  0.27 (0.93)  0.502
  NP_958781 (mean (SD))  0.44 (1.14)  0.27 (0.93)  0.499
  NP_958780 (mean (SD))  0.44 (1.14)  0.27 (0.93)  0.496
  NP_958783 (mean (SD))  0.44 (1.14)  0.27 (0.93)  0.495
  NP_958784 (mean (SD))  0.44 (1.14)  0.27 (0.93)  0.495
  NP_112598 (mean (SD)) -0.20 (2.28) -0.36 (1.97)  0.748
  NP_001611 (mean (SD)) -0.57 (1.54)  0.84 (1.19) <0.001
```

Using tableone

```
CreateTableOne(
  data = brca %>% filter(!is.na(ER.Status)),
  vars = brca %>%
    select(starts_with('NP')) %>%
    names(),
  strata = 'ER.Status',
  test = T,
  testNormal = t.test,
  argsNormal = list(var.equal=F)
)
```

```
                     Stratified by ER.Status
                      Negative     Positive     p      test
  n                      38           69
  NP_958782 (mean (SD))  0.43 (1.13)  0.27 (0.93)  0.529
  NP_958785 (mean (SD))  0.44 (1.14)  0.27 (0.93)  0.524
  NP_958786 (mean (SD))  0.44 (1.14)  0.27 (0.93)  0.520
  NP_000436 (mean (SD))  0.43 (1.14)  0.27 (0.93)  0.534
  NP_958781 (mean (SD))  0.44 (1.14)  0.27 (0.93)  0.530
  NP_958780 (mean (SD))  0.44 (1.14)  0.27 (0.93)  0.528
  NP_958783 (mean (SD))  0.44 (1.14)  0.27 (0.93)  0.527
  NP_958784 (mean (SD))  0.44 (1.14)  0.27 (0.93)  0.527
  NP_112598 (mean (SD)) -0.20 (2.28) -0.36 (1.97)  0.761
  NP_001611 (mean (SD)) -0.57 (1.54)  0.84 (1.19) <0.001
```

Tests for discrete data

Testing whether the distribution of a categorical variable differs by levels of another categorical variable can be done using either the Chi-square test (`chisq.test`) or the Fisher's test (`fisher.test`). Both require you to create a 2x2 table first.

```
fisher.test(table(brca$Tumor, brca$ER.Status))
```

```

    Fisher's Exact Test for Count Data

data:  table(brca$Tumor, brca$ER.Status)
p-value = 0.6003
alternative hypothesis: two.sided
```

Tests for discrete data

Testing whether the distribution of a categorical variable differs by levels of another categorical variable can be done using either the Chi-square test (`chisq.test`) or the Fisher's test (`fisher.test`). Both require you to create a 2x2 table first.

```
chisq.test(table(brca$Tumor, brca$ER.Status))
```

```

    Pearson's Chi-squared test

data:  table(brca$Tumor, brca$ER.Status)
X-squared = 2.094, df = 3, p-value = 0.5531
```

Tests for discrete data

We can use `broom::tidy` for either of these

```
chisq.test(table(brca$Tumor, brca$ER.Status)) %>%
  broom::tidy()
```

```
# A tibble: 1 x 4
  statistic p.value parameter method
      <dbl>   <dbl>     <int> <chr>
1      2.09   0.553         3 Pearson's Chi-squared test
```

Using tableone

```
CreateCatTable(vars = c('Tumor','Node','Metastasis'),
               data = filter(brca, !is.na(ER.Status)),
               strata = 'ER.Status',
               test = T) # chisq.test
```

```
                     Stratified by ER.Status
                      Negative   Positive   p      test
  n                   38         69
  Tumor (%)                                 0.553
     T1                6 (15.8)  10 (14.5)
     T2               26 (68.4)  40 (58.0)
     T3                5 (13.2)  14 (20.3)
     T4                1 ( 2.6)   5 ( 7.2)
  Node (%)                                  0.685
     N0               22 (57.9)  32 (46.4)
     N1                8 (21.1)  21 (30.4)
     N2                5 (13.2)  10 (14.5)
     N3                3 ( 7.9)   6 ( 8.7)
  Metastasis = M1 (%)  1 ( 2.6)   1 ( 1.4)   1.000
```

Using tableone

```
c1 <- CreateCatTable(vars = c('Tumor','Node','Metastasis'),
                     data = filter(brca, !is.na(ER.Status)),
                     strata = 'ER.Status',
                     test = T)
print(c1, exact = c('Tumor','Node','Metastasis')) # fisher.test
```

```
                     Stratified by ER.Status
                      Negative   Positive   p      test
  n                   38         69
  Tumor (%)                                 0.600 exact
     T1                6 (15.8)  10 (14.5)
     T2               26 (68.4)  40 (58.0)
     T3                5 (13.2)  14 (20.3)
     T4                1 ( 2.6)   5 ( 7.2)
  Node (%)                                  0.695 exact
     N0               22 (57.9)  32 (46.4)
     N1                8 (21.1)  21 (30.4)
     N2                5 (13.2)  10 (14.5)
     N3                3 ( 7.9)   6 ( 8.7)
  Metastasis = M1 (%)  1 ( 2.6)   1 ( 1.4)   1.000 exact
```