

Intro to R

Data Summarization

Recap

- `select()`: subset and/or reorder columns
- `filter()`: remove rows
- `arrange()`: reorder rows
- `mutate()`: create new columns or modify them
- `select()` and `filter()` can be combined together
- remove a column: `select()` with negative sign (`-col_name`)
- you can do sequential steps: especially using pipes `%>%`

[Cheatsheet](#)

Another Cheatsheet

<https://raw.githubusercontent.com/rstudio/cheatsheets/main/data-transformation.pdf>

Data transformation with dplyr : : CHEAT SHEET



dplyr functions work with pipes and expect **tidy data**. In tidy data:



&



pipes

Each **variable** is in its own **column**

Each **observation**, or **case**, is in its own **row**

$x \%>\% f(y)$ becomes $f(x, y)$

Summarise Cases

Apply **summary functions** to columns to create a new table of summary statistics. Summary functions take vectors as input and return one value (see back).

summary function



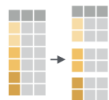
summarise(.data, ...)
Compute table of summaries.
`summarise(mtcars, avg = mean(mpg))`



count(.data, ..., wt = NULL, sort = FALSE, name = NULL) Count number of rows in each group defined by the variables in ... Also **tally()**.
`count(mtcars, cyl)`

Group Cases

Use **group_by(.data, ..., .add = FALSE, .drop = TRUE)** to create a "grouped" copy of a table grouped by columns in ... dplyr functions will manipulate each "group" separately and combine the results.



`mtcars %>%
group_by(cyl) %>%
summarise(avg = mean(mpg))`

Manipulate Cases

EXTRACT CASES

Row functions return a subset of rows as a new table.



filter(.data, ..., .preserve = FALSE) Extract rows that meet logical criteria.
`filter(mtcars, mpg > 20)`



distinct(.data, ..., .keep_all = FALSE) Remove rows with duplicate values.
`distinct(mtcars, gear)`



slice(.data, ..., .preserve = FALSE) Select rows by position.
`slice(mtcars, 10:15)`



slice_sample(.data, ..., n, prop, weight_by = NULL, replace = FALSE) Randomly select rows. Use `n` to select a number of rows and `prop` to select a fraction of rows.
`slice_sample(mtcars, n = 5, replace = TRUE)`



slice_min(.data, order_by, ..., n, prop, with_ties = TRUE) and **slice_max()** Select rows with the lowest and highest values.
`slice_min(mtcars, mpg, prop = 0.25)`

slice_head(.data, ..., n, prop) and **slice_tail()** Select the first or last rows.
`slice_head(mtcars, n = 5)`

Logical and boolean operators to use with filter()

<code>==</code>	<code><</code>	<code><=</code>	<code>is.na()</code>	<code>%in%</code>	<code> </code>	<code>xor()</code>
<code>!=</code>	<code>></code>	<code>>=</code>	<code>!is.na()</code>	<code>!</code>	<code>&</code>	

See **?base::Logic** and **?Comparison** for help.

ARRANGE CASES

Manipulate Variables

EXTRACT VARIABLES

Column functions return a set of columns as a new vector or table.



pull(.data, var = -1, name = NULL, ...) Extract column values as a vector, by name or index.
`pull(mtcars, wt)`



select(.data, ...) Extract columns as a table.
`select(mtcars, mpg, wt)`



relocate(.data, ..., .before = NULL, .after = NULL) Move columns to new position.
`relocate(mtcars, mpg, cyl, .after = last_col())`

Use these helpers with select() and across()

e.g. `select(mtcars, mpg:cyl)`

contains(match)	num_range(prefix, range)	• e.g. <code>mpg:cyl</code>
ends_with(match)	all_of(x)/any_of(x, ..., vars)	• e.g. <code>-gear</code>
starts_with(match)	matches(match)	everything()

MANIPULATE MULTIPLE VARIABLES AT ONCE



across(.cols, .funs, ..., .names = NULL) Summarise or mutate multiple columns in the same way.
`summarise(mtcars, across(everything(), mean))`



c_across(.cols) Compute across columns in row-wise data.
`transmute(rowwise(UKgas), total = sum(c_across(1:2)))`

MAKE NEW VARIABLES

Apply **vectorized functions** to columns. Vectorized functions take vectors as input and return vectors of the same length as output (see back).

Data Summarization

- Basic statistical summarization
 - `mean(x)`: takes the mean of `x`
 - `sd(x)`: takes the standard deviation of `x`
 - `median(x)`: takes the median of `x`
 - `quantile(x)`: displays sample quantiles of `x`. Default is min, IQR, max
 - `range(x)`: displays the range. Same as `c(min(x), max(x))`
 - `sum(x)`: sum of `x`
 - `max(x)`: maximum value in `x`
 - `min(x)`: minimum value in `x`
- **all have the `na.rm` = argument for missing data**

Statistical summarization

The vector getting summarized goes inside the parentheses:

```
x <- c(1, 5, 7, 4, 2, 8)  
mean(x)
```

```
[1] 4.5
```

```
range(x)
```

```
[1] 1 8
```

```
sum(x)
```

```
[1] 27
```

Statistical summarization

Note that many of these functions have additional inputs regarding missing data, typically requiring the `na.rm` argument (“remove NAs”).

```
x <- c(1, 5, 7, 4, 2, 8, NA)
mean(x)
```

```
[1] NA
```

```
mean(x, na.rm = TRUE)
```

```
[1] 4.5
```

```
quantile(x)
```

```
Error in quantile.default(x): missing values and NaN's not allowed if 'na.rm' is FALSE
```

```
quantile(x, na.rm = TRUE)
```

0%	25%	50%	75%	100%
1.0	2.5	4.5	6.5	8.0

Statistical summarization

We will talk more about data types later, but you can only do summarization on numeric or logical types. Not characters.

```
x <- c(1, 5, 7, 4, 2, 8)
sum(x)
```

```
[1] 27
```

```
y <- c(TRUE, FALSE, FALSE, TRUE) # FALSE == 0 and TRUE == 1
sum(y)
```

```
[1] 2
```

```
z <- c("TRUE", "FALSE", "FALSE", "TRUE")
sum(z)
```

```
Error in sum(z): invalid 'type' (character) of argument
```

Some examples

We can use the `jhu_cars` to explore different ways of summarizing data. The `head` command displays the first rows of an object:

```
library(jhur)
head(jhu_cars)
```

	car	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
1	Mazda RX4	21.0	6	160	110	3.90	2.620	16.46	0	1	4	4
2	Mazda RX4 Wag	21.0	6	160	110	3.90	2.875	17.02	0	1	4	4
3	Datsun 710	22.8	4	108	93	3.85	2.320	18.61	1	1	4	1
4	Hornet 4 Drive	21.4	6	258	110	3.08	3.215	19.44	1	0	3	1
5	Hornet Sportabout	18.7	8	360	175	3.15	3.440	17.02	0	0	3	2
6	Valiant	18.1	6	225	105	2.76	3.460	20.22	1	0	3	1

Statistical summarization

You might see base R `$` to reference/select columns from a `data.frame/tibble`:

```
mean(jhu_cars$hp)
```

```
[1] 146.6875
```

```
quantile(jhu_cars$hp)
```

0%	25%	50%	75%	100%
52.0	96.5	123.0	180.0	335.0

Statistical summarization

The “tidy” way:

```
jhu_cars %>% pull(hp) %>% mean() # alt: pull(jhu_cars, hp) %>% mean()
```

```
[1] 146.6875
```

```
jhu_cars %>% pull(hp) %>% quantile()
```

0%	25%	50%	75%	100%
52.0	96.5	123.0	180.0	335.0

Statistical summarization

```
jhu_cars %>% pull(wt) %>% median()
```

```
[1] 3.325
```

```
jhu_cars %>% pull(wt) %>% quantile(probs = 0.6)
```

```
60%  
3.44
```

Data Summarization on data frames

- Basic statistical summarization
 - `rowMeans(x)`: takes the means of each row of `x`
 - `colMeans(x)`: takes the means of each column of `x`
 - `rowSums(x)`: takes the sum of each row of `x`
 - `colSums(x)`: takes the sum of each column of `x`
 - `summary(x)`: for data frames, displays the quantile information

TB Incidence

Let's read in a `tibble` of values from TB incidence.

If you have the `jhur` package installed successfully:

```
library(jhur)
tb <- read_tb()
```

If not, download the `xlsx` file from

http://jhudatascience.org/intro_to_r/data/tb_incidence.xlsx and read it in:

```
library(readxl)
tb <- read_excel("tb_incidence.xlsx")
```

TB Incidence

Check out the data:

```
head(tb)
```

```
# A tibble: 6 × 19
  `TB incidence, all f...` `1990` `1991` `1992` `1993` `1994` `1995` `1996` `1997`
  <chr>                  <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1 Afghanistan           168    168    168    168    168    168    168    168
2 Albania                25     24     25     26     26     27     27     28
3 Algeria                38     38     39     40     41     42     43     44
4 American Samoa         21      7      2      9      9     11      0     12
5 Andorra                36     34     32     30     29     27     26     26
6 Angola                 205    209    214    218    222    226    231    236
# ... with 10 more variables: `1998` <dbl>, `1999` <dbl>, `2000` <dbl>,
#   `2001` <dbl>, `2002` <dbl>, `2003` <dbl>, `2004` <dbl>, `2005` <dbl>,
#   `2006` <dbl>, `2007` <dbl>
```

TB Incidence

Check out the data:

```
str(tb)
```

```
tibble [208 × 19] (S3: tbl_df/tbl/data.frame)
 $ TB incidence, all forms (per 100 000 population per year): chr [1:208] "Afghanistan" "Albania"
 $ 1990 : num [1:208] 168 25 38 21 36 205 2
 $ 1991 : num [1:208] 168 24 38 7 34 209 24
 $ 1992 : num [1:208] 168 25 39 2 32 214 24
 $ 1993 : num [1:208] 168 26 40 9 30 218 24
 $ 1994 : num [1:208] 168 26 41 9 29 222 23
 $ 1995 : num [1:208] 168 27 42 11 27 226 2
 $ 1996 : num [1:208] 168 27 43 0 26 231 23
 $ 1997 : num [1:208] 168 28 44 12 26 236 2
 $ 1998 : num [1:208] 168 28 46 6 25 240 23
 $ 1999 : num [1:208] 168 27 47 8 23 245 23
 $ 2000 : num [1:208] 168 25 48 6 22 250 23
 $ 2001 : num [1:208] 168 23 49 6 21 255 22
 $ 2002 : num [1:208] 168 23 50 4 21 260 22
 $ 2003 : num [1:208] 168 22 51 5 20 265 22
 $ 2004 : num [1:208] 168 21 53 9 20 270 22
 $ 2005 : num [1:208] 168 20 54 10 19 276 2
 $ 2006 : num [1:208] 168 18 55 7 19 281 22
 $ 2007 : num [1:208] 168 17 57 5 19 287 22
```

Indicator of TB

Before we go further, let's rename the first column using the `rename()` function in `dplyr`.

In this case, we have to use the backticks (```) because there are spaces and funky characters in the name:

```
library(dplyr)
tb <- tb %>% rename(country = `TB incidence, all forms (per 100 000 population per year)`)
```


Indicator of TB

`colnames()` will show us the column names and show that country is renamed:

```
colnames(tb)
```

```
[1] "country" "1990"    "1991"    "1992"    "1993"    "1994"    "1995"  
[8] "1996"    "1997"    "1998"    "1999"    "2000"    "2001"    "2002"  
[15] "2003"    "2004"    "2005"    "2006"    "2007"
```

Summarize the data: **dplyr** `summarize()` function

`summarize` creates a summary table of a column you're interested in.

General format - Not the code!

`{data to use} %>%`

`summarize({summary column name} = {operator(source column)})`

Summarize the data: `dplyr summarize()` function

`summarize` creates a summary table of a column you're interested in.

General format - Not the code!

```
{data to use} %>%  
  summarize({summary column name} = {operator(source column)})
```

```
tb %>%  
  summarize(mean_1991 = mean(`1991`))
```

```
# A tibble: 1 × 1  
  mean_1991  
    <dbl>  
1         NA
```

```
tb %>%  
  summarize(mean_1991 = mean(`1991`, na.rm = TRUE))
```

```
# A tibble: 1 × 1  
  mean_1991  
    <dbl>  
1      108.
```

Summarize the data: **dplyr** `summarize()` function

`summarize()` can do multiple operations at once. Just separate by a comma.

```
tb %>%
  summarize(mean_1991 = mean(`1991`, na.rm = TRUE),
            median_1991 = median(`1991`, na.rm = TRUE),
            median(`2000`, na.rm = TRUE))

# A tibble: 1 × 3
  mean_1991 median_1991 `median(\`2000\`, na.rm = TRUE)`
    <dbl>         <dbl>                <dbl>
1    108.          58                60
```

Notice how when we forget to provide a new name, output is still provided, but the column name is messy.

Summarize the data: `dplyr summarize()` function

This looks better.

```
tb %>%  
  summarize(mean_1991 = mean(`1991`, na.rm = TRUE),  
            median_1991 = median(`1991`, na.rm = TRUE),  
            median_2000 = median(`2000`, na.rm = TRUE))
```

```
# A tibble: 1 × 3  
  mean_1991 median_1991 median_2000  
    <dbl>      <dbl>      <dbl>  
1    108.         58         60
```

Row means

`colMeans()` and `rowMeans()` require **all numeric data**.

Let's see what the mean is across each row (country):

```
tb_2 <- column_to_rownames(tb, var = "country") # opposite of rownames_to_column() !
head(tb_2, n = 2)
```

	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999	2000	2001	2002
Afghanistan	168	168	168	168	168	168	168	168	168	168	168	168	168
Albania	25	24	25	26	26	27	27	28	28	27	25	23	23
	2003	2004	2005	2006	2007								
Afghanistan	168	168	168	168	168								
Albania	22	21	20	18	17								

```
rowMeans(tb_2, na.rm = TRUE)
```

Afghanistan	Albania
168.000000	24.000000
Algeria	American Samoa
46.388889	7.611111
Andorra	Angola
24.944444	243.888889
Anguilla	Antigua and Barbuda
22.833333	7.222222
Argentina	Armenia
43.666667	57.611111
Australia	Austria
6.444444	16.333333
Azerbaijan	Bahamas

Column means

`colMeans()` and `rowMeans()` require **all numeric data**.

Let's see what the mean is across each column (year):

```
colMeans(tb_2, na.rm = TRUE)
```

1990	1991	1992	1993	1994	1995	1996	1997
105.5797	107.6715	108.3140	110.3188	111.9662	114.1981	115.3527	118.8792
1998	1999	2000	2001	2002	2003	2004	2005
121.5169	125.0435	127.8454	130.7488	136.1739	136.1932	136.9662	135.6683
2006	2007						
134.6106	133.3865						

summary() Function

Using `summary()` can give you rough snapshots of each numeric column (character columns are skipped):

```
summary(tb)
```

country	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999	2000	2001
Length:208	Min. : 0.0	Min. : 4.0	Min. : 2.0	Min. : 4.0	Min. : 0	Min. : 3.0	Min. : 0.0	Min. : 0.0	Min. : 0.0	Min. : 0.0	Min. : 0.0	Min. : 0.0
Class :character	1st Qu.: 27.5	1st Qu.: 27.0	1st Qu.: 27.0	1st Qu.: 27.5	1st Qu.: 26	1st Qu.: 26.5	1st Qu.: 25.5	1st Qu.: 24.5	1st Qu.: 23.5	1st Qu.: 22.5	1st Qu.: 21.5	1st Qu.: 19.0
Mode :character	Median : 60.0	Median : 58.0	Median : 56.0	Median : 56.0	Median : 57	Median : 58.0	Median : 60.0	Median : 64.0	Median : 63.0	Median : 66.0	Median : 60.0	Median : 59.0
	Mean :105.6	Mean :107.7	Mean :108.3	Mean :110.3	Mean :112	Mean :114.2	Mean :115.4	Mean :118.9	Mean :121.5	Mean :125.0	Mean :127.8	Mean :130.7
	3rd Qu.:165.0	3rd Qu.:171.0	3rd Qu.:171.5	3rd Qu.:171.0	3rd Qu.:174	3rd Qu.:177.5	3rd Qu.:179.0	3rd Qu.:181.0	3rd Qu.:188.5	3rd Qu.:192.5	3rd Qu.:191.0	3rd Qu.:189.5
	Max. :585.0	Max. :594.0	Max. :606.0	Max. :618.0	Max. :630	Max. :642.0	Max. :655.0	Max. :668.0	Max. :681.0	Max. :695.0	Max. :801.0	Max. :916.0
	NA's :1	NA's :1	NA's :1	NA's :1	NA's :1	NA's :1	NA's :1	NA's :1	NA's :1	NA's :1	NA's :1	NA's :1

Summary & Lab Part 1

- summary stats (`mean()`) work with `pull()`
- don't forget the `na.rm = TRUE` argument!
- `summary(x)`: quantile information
- `summarize`: creates a summary table of columns of interest

[Class Website](#)

[Lab](#)

Youth Tobacco Survey

Here we will be using the Youth Tobacco Survey data:

http://jhudatascience.org/intro_to_r/data/Youth_Tobacco_Survey_YTS_Data.csv

```
yts <- read_yts()
head(yts)
```

```
# A tibble: 6 × 31
  YEAR LocationAbbr LocationDesc TopicType TopicDesc MeasureDesc DataSource
<dbl> <chr>         <chr>         <chr>         <chr>         <chr>         <chr>
1  2015 AZ          Arizona      Tobacco Use ... Cessatio... Percent of... YTS
2  2015 AZ          Arizona      Tobacco Use ... Cessatio... Percent of... YTS
3  2015 AZ          Arizona      Tobacco Use ... Cessatio... Percent of... YTS
4  2015 AZ          Arizona      Tobacco Use ... Cessatio... Quit Attem... YTS
5  2015 AZ          Arizona      Tobacco Use ... Cessatio... Quit Attem... YTS
6  2015 AZ          Arizona      Tobacco Use ... Cessatio... Quit Attem... YTS
# ... with 24 more variables: Response <chr>, Data_Value_Unit <chr>,
#   Data_Value_Type <chr>, Data_Value <dbl>, Data_Value_Footnote_Symbol <chr>,
#   Data_Value_Footnote <chr>, Data_Value_Std_Err <dbl>,
#   Low_Confidence_Limit <dbl>, High_Confidence_Limit <dbl>, Sample_Size <dbl>,
#   Gender <chr>, Race <chr>, Age <chr>, Education <chr>, GeoLocation <chr>,
#   TopicTypeId <chr>, TopicId <chr>, MeasureId <chr>, StratificationID1 <chr>,
#   StratificationID2 <chr>, StratificationID3 <chr>, ...
```

Column to vector

Let's work with one column as a vector using `pull()`.

```
locations <- yts %>% pull(LocationDesc)  
locations
```

```
[1] "Arizona"  
[3] "Arizona"  
[5] "Arizona"  
[7] "Arizona"  
[9] "Arizona"  
[11] "Arizona"  
[13] "Arizona"  
[15] "Arizona"  
[17] "Arizona"  
[19] "Arizona"  
[21] "Arizona"  
[23] "Arizona"  
[25] "Connecticut"  
[27] "Connecticut"  
[29] "Connecticut"  
[31] "Connecticut"  
[33] "Connecticut"  
[35] "Connecticut"  
[37] "Connecticut"  
[39] "Connecticut"  
[41] "Connecticut"  
[43] "Connecticut"  
[45] "Connecticut"  
[47] "Connecticut"  
[49] "Connecticut"
```

Length and unique

`unique(x)` will return the unique elements of `x`

```
unique(locations)
```

```
[1] "Arizona"
[3] "Georgia"
[5] "Illinois"
[7] "Mississippi"
[9] "Missouri"
[11] "Nebraska"
[13] "North Carolina"
[15] "Pennsylvania"
[17] "West Virginia"
[19] "Delaware"
[21] "Guam"
[23] "Indiana"
[25] "Oklahoma"
[27] "Michigan"
[29] "Arkansas"
[31] "Iowa"
[33] "Virginia"
[35] "Rhode Island"
[37] "Tennessee"
[39] "Virgin Islands"
[41] "Idaho"
[43] "Maryland"
[45] "New York"
[47] "Colorado"
[49] "Texas"

"Connecticut"
"Hawaii"
"Louisiana"
"Utah"
"National (States and DC)"
"New Jersey"
"North Dakota"
"South Carolina"
"Alabama"
"Minnesota"
"Ohio"
"Kansas"
"Wisconsin"
"New Hampshire"
"Kentucky"
"South Dakota"
"Puerto Rico"
"New Mexico"
"Vermont"
"California"
"Florida"
"Massachusetts"
"Maine"
"District of Columbia"
"Wyoming"
```

Length and unique

`length` will tell you the length of a vector. Combined with `unique`, tells you the number of unique elements:

```
length(unique(locations))
```

```
[1] 50
```

unique() and distinct()

These functions work similarly, but expect different types of objects

```
unique(locations) # vector
```

```
[1] "Arizona"           "Connecticut"
[3] "Georgia"           "Hawaii"
[5] "Illinois"           "Louisiana"
[7] "Mississippi"        "Utah"
[9] "Missouri"           "National (States and DC)"
[11] "Nebraska"           "New Jersey"
[13] "North Carolina"     "North Dakota"
[15] "Pennsylvania"
[ reached getOption("max.print") -- omitted 35 entries ]
```

```
yts %>% distinct(LocationDesc) # tibble / data frame
```

```
# A tibble: 50 × 1
  LocationDesc
  <chr>
1 Arizona
2 Connecticut
3 Georgia
4 Hawaii
5 Illinois
6 Louisiana
7 Mississippi
8 Utah
9 Missouri
10 National (States and DC)
# ... with 40 more rows
```

dplyr: count

Use `count` to return a frequency table of unique elements of a data.frame.

```
yts %>% count(LocationDesc)
```

```
# A tibble: 50 × 2
```

	LocationDesc	n
	<chr>	<int>
1	Alabama	378
2	Arizona	240
3	Arkansas	210
4	California	96
5	Colorado	48
6	Connecticut	384
7	Delaware	312
8	District of Columbia	48
9	Florida	96
10	Georgia	282

```
# ... with 40 more rows
```

dp1yr: count

Multiple columns listed further subdivides the count.

```
yts %>% count(LocationDesc, TopicDesc)
```

```
# A tibble: 146 × 3
  LocationDesc TopicDesc      n
  <chr>        <chr>    <int>
1 Alabama      Cessation (Youth)      90
2 Alabama      Cigarette Use (Youth) 144
3 Alabama      Smokeless Tobacco Use (Youth) 144
4 Arizona      Cessation (Youth)      60
5 Arizona      Cigarette Use (Youth)   99
6 Arizona      Smokeless Tobacco Use (Youth) 81
7 Arkansas      Cessation (Youth)      42
8 Arkansas      Cigarette Use (Youth)   78
9 Arkansas      Smokeless Tobacco Use (Youth) 90
10 California   Cessation (Youth)      24
# ... with 136 more rows
```


dp1yr: count

Multiple columns listed further subdivides the count.

```
yts %>% count(LocationDesc, TopicDesc)
```

```
# A tibble: 146 × 3
  LocationDesc TopicDesc      n
  <chr>        <chr>    <int>
1 Alabama      Cessation (Youth)      90
2 Alabama      Cigarette Use (Youth) 144
3 Alabama      Smokeless Tobacco Use (Youth) 144
4 Arizona      Cessation (Youth)      60
5 Arizona      Cigarette Use (Youth)   99
6 Arizona      Smokeless Tobacco Use (Youth) 81
7 Arkansas     Cessation (Youth)      42
8 Arkansas     Cigarette Use (Youth)   78
9 Arkansas     Smokeless Tobacco Use (Youth) 90
10 California  Cessation (Youth)      24
# ... with 136 more rows
```

Note: count() includes NAs

Grouping

Perform Operations By Groups: dplyr

`group_by` allows you group the data set by variables/columns you specify:

```
# Regular data  
yts
```

```
# A tibble: 9,794 × 31  
  YEAR LocationAbbr LocationDesc TopicType TopicDesc MeasureDesc DataSource  
  <dbl> <chr>      <chr>      <chr>      <chr>      <chr>      <chr>  
1  2015 AZ        Arizona    Tobacco Use... Cessatio... Percent of... YTS  
2  2015 AZ        Arizona    Tobacco Use... Cessatio... Percent of... YTS  
3  2015 AZ        Arizona    Tobacco Use... Cessatio... Percent of... YTS  
4  2015 AZ        Arizona    Tobacco Use... Cessatio... Quit Attem... YTS  
5  2015 AZ        Arizona    Tobacco Use... Cessatio... Quit Attem... YTS  
6  2015 AZ        Arizona    Tobacco Use... Cessatio... Quit Attem... YTS  
7  2015 AZ        Arizona    Tobacco Use... Cigarett... Smoking St... YTS  
8  2015 AZ        Arizona    Tobacco Use... Cigarett... Smoking St... YTS  
9  2015 AZ        Arizona    Tobacco Use... Cigarett... Smoking St... YTS  
10 2015 AZ        Arizona    Tobacco Use... Cigarett... Smoking St... YTS  
# ... with 9,784 more rows, and 24 more variables: Response <chr>,  
# Data_Value_Unit <chr>, Data_Value_Type <chr>, Data_Value <dbl>,  
# Data_Value_Footnote_Symbol <chr>, Data_Value_Footnote <chr>,  
# Data_Value_Std_Err <dbl>, Low_Confidence_Limit <dbl>,  
# High_Confidence_Limit <dbl>, Sample_Size <dbl>, Gender <chr>, Race <chr>,  
# Age <chr>, Education <chr>, GeoLocation <chr>, TopicTypeId <chr>,  
# TopicId <chr>, MeasureId <chr>, StratificationID1 <chr>, ...
```

Perform Operations By Groups: dplyr

`group_by` allows you group the data set by variables/columns you specify:

```
yts_grouped <- yts %>% group_by(Response)
yts_grouped
```

```
# A tibble: 9,794 × 31
```

```
# Groups:   Response [4]
```

	YEAR	LocationAbbr	LocationDesc	TopicType	TopicDesc	MeasureDesc	DataSource
	<dbl>	<chr>	<chr>	<chr>	<chr>	<chr>	<chr>
1	2015	AZ	Arizona	Tobacco Use...	Cessatio...	Percent of...	YTS
2	2015	AZ	Arizona	Tobacco Use...	Cessatio...	Percent of...	YTS
3	2015	AZ	Arizona	Tobacco Use...	Cessatio...	Percent of...	YTS
4	2015	AZ	Arizona	Tobacco Use...	Cessatio...	Quit Attem...	YTS
5	2015	AZ	Arizona	Tobacco Use...	Cessatio...	Quit Attem...	YTS
6	2015	AZ	Arizona	Tobacco Use...	Cessatio...	Quit Attem...	YTS
7	2015	AZ	Arizona	Tobacco Use...	Cigarett...	Smoking St...	YTS
8	2015	AZ	Arizona	Tobacco Use...	Cigarett...	Smoking St...	YTS
9	2015	AZ	Arizona	Tobacco Use...	Cigarett...	Smoking St...	YTS
10	2015	AZ	Arizona	Tobacco Use...	Cigarett...	Smoking St...	YTS

```
# ... with 9,784 more rows, and 24 more variables: Response <chr>,
# Data_Value_Unit <chr>, Data_Value_Type <chr>, Data_Value <dbl>,
# Data_Value_Footnote_Symbol <chr>, Data_Value_Footnote <chr>,
# Data_Value_Std_Err <dbl>, Low_Confidence_Limit <dbl>,
# High_Confidence_Limit <dbl>, Sample_Size <dbl>, Gender <chr>, Race <chr>,
# Age <chr>, Education <chr>, GeoLocation <chr>, TopicTypeId <chr>,
# TopicId <chr>, MeasureId <chr>, StratificationID1 <chr>, ...
```

Summarize the grouped data

It's grouped! Grouping doesn't change the data in any way, but how **functions operate on it**. Now we can summarize `Data_Value` (percent of respondents) by group:

```
yts_grouped %>% summarize(avg_percent = mean(Data_Value, na.rm = TRUE))
```

```
# A tibble: 4 × 2
  Response avg_percent
  <chr>      <dbl>
1 Current    9.68
2 Ever     26.1
3 Frequent   3.48
4 <NA>      53.5
```

Use the **pipe** to string these together!

Pipe yts into group_by, then pipe that into summarize:

```
yls %>%  
  group_by(Response) %>%  
  summarize(avg_percent = mean(Data_Value, na.rm = TRUE),  
            max_percent = max(Data_Value, na.rm = TRUE))
```

A tibble: 4 × 3

	Response	avg_percent	max_percent
	<chr>	<dbl>	<dbl>
1	Current	9.68	40.6
2	Ever	26.1	98
3	Frequent	3.48	23.9
4	<NA>	53.5	81.9

Group by as many variables as you want

group_by Response and Education:

```
yts %>%  
  group_by(Response, Education) %>%  
  summarize(avg_percent = mean(Data_Value, na.rm = TRUE),  
            max_percent = max(Data_Value, na.rm = TRUE))
```

```
# A tibble: 8 × 4  
# Groups:   Response [4]  
  Response Education      avg_percent max_percent  
  <chr>      <chr>          <dbl>         <dbl>  
1 Current  High School      14.1          40.6  
2 Current  Middle School    5.73          26.1  
3 Ever     High School     34.7          96.2  
4 Ever     Middle School   18.6           98  
5 Frequent High School     5.91         23.9  
6 Frequent Middle School  1.33           8  
7 <NA>     High School     53.8         78.9  
8 <NA>     Middle School   53.2         81.9
```

Ungroup the data

The `ungroup` function will allow you to clear the groups from the data. You can also overwrite the first `group_by` with a new one.

```
yts <- ungroup(yts)
yts
```

```
# A tibble: 9,794 × 31
  YEAR LocationAbbr LocationDesc TopicType TopicDesc MeasureDesc DataSource
<dbl> <chr>         <chr>         <chr>      <chr>      <chr>      <chr>
1  2015 AZ          Arizona      Tobacco Use... Cessatio... Percent of... YTS
2  2015 AZ          Arizona      Tobacco Use... Cessatio... Percent of... YTS
3  2015 AZ          Arizona      Tobacco Use... Cessatio... Percent of... YTS
4  2015 AZ          Arizona      Tobacco Use... Cessatio... Quit Attem... YTS
5  2015 AZ          Arizona      Tobacco Use... Cessatio... Quit Attem... YTS
6  2015 AZ          Arizona      Tobacco Use... Cessatio... Quit Attem... YTS
7  2015 AZ          Arizona      Tobacco Use... Cigarett... Smoking St... YTS
8  2015 AZ          Arizona      Tobacco Use... Cigarett... Smoking St... YTS
9  2015 AZ          Arizona      Tobacco Use... Cigarett... Smoking St... YTS
10 2015 AZ          Arizona      Tobacco Use... Cigarett... Smoking St... YTS
# ... with 9,784 more rows, and 24 more variables: Response <chr>,
# Data_Value_Unit <chr>, Data_Value_Type <chr>, Data_Value <dbl>,
# Data_Value_Footnote_Symbol <chr>, Data_Value_Footnote <chr>,
# Data_Value_Std_Err <dbl>, Low_Confidence_Limit <dbl>,
# High_Confidence_Limit <dbl>, Sample_Size <dbl>, Gender <chr>, Race <chr>,
# Age <chr>, Education <chr>, GeoLocation <chr>, TopicTypeId <chr>,
# TopicId <chr>, MeasureId <chr>, StratificationID1 <chr>, ...
```


group_by with mutate - just add data

We can also use `mutate` to calculate the mean value for each year and add it as a column:

```
yts %>%  
  group_by(YEAR) %>%  
  mutate(year_avg = mean(Data_Value, na.rm = TRUE)) %>%  
  select(LocationDesc, Data_Value, year_avg)
```

```
# A tibble: 9,794 × 4
```

```
# Groups:   YEAR [17]
```

	YEAR	LocationDesc	Data_Value	year_avg
	<dbl>	<chr>	<dbl>	<dbl>
1	2015	Arizona	NA	15.2
2	2015	Arizona	NA	15.2
3	2015	Arizona	NA	15.2
4	2015	Arizona	NA	15.2
5	2015	Arizona	NA	15.2
6	2015	Arizona	NA	15.2
7	2015	Arizona	3.2	15.2
8	2015	Arizona	3.2	15.2
9	2015	Arizona	3.1	15.2
10	2015	Arizona	12.5	15.2

```
# ... with 9,784 more rows
```

Counting

There are other functions, such as `n()` count the number of observations (NAs included).

```
yts %>%  
  group_by(YEAR) %>%  
  summarize(n = n(),  
            mean = mean(Data_Value, na.rm = TRUE))
```

```
# A tibble: 17 × 3  
  YEAR      n  mean  
  <dbl> <int> <dbl>  
1  1999   372  26.1  
2  2000  1224  26.7  
3  2001   426  23.4  
4  2002  1016  25.2  
5  2003   498  21.3  
6  2004   611  20.7  
7  2005   636  21.8  
8  2006   518  21.8  
9  2007   516  20.0  
10 2008   483  18.2  
11 2009   686  18.3  
12 2010   447  17.8  
13 2011   521  17.8  
14 2012   244  15.5  
15 2013   685  16.7  
16 2014   334  15.7  
17 2015   577  15.2
```

Counting

`count()` and `n()` can give very similar information.

```
mtcars %>% count(cyl)
```

```
  cyl  n
1    4 11
2    6  7
3    8 14
```

```
mtcars %>% group_by(cyl) %>% summarize(n()) # n() typically used with summarize
```

```
# A tibble: 3 × 2
```

```
  cyl `n()`
<dbl> <int>
1     4    11
2     6     7
3     8    14
```

Summary & Lab Part 2

- `count(x)`: what unique values do you have?
 - `pull()` gives a vector
 - `unique()` combined with `length()`
- `group_by()`: changes all subsequent functions
 - combine with `summarize()` to get statistics per group
 - combine with `mutate()` to add column
- `summarize()` with `n()` gives the count (NAs included)

[Class Website](#)

[Lab](#)

Preview: plotting

Basic Plots

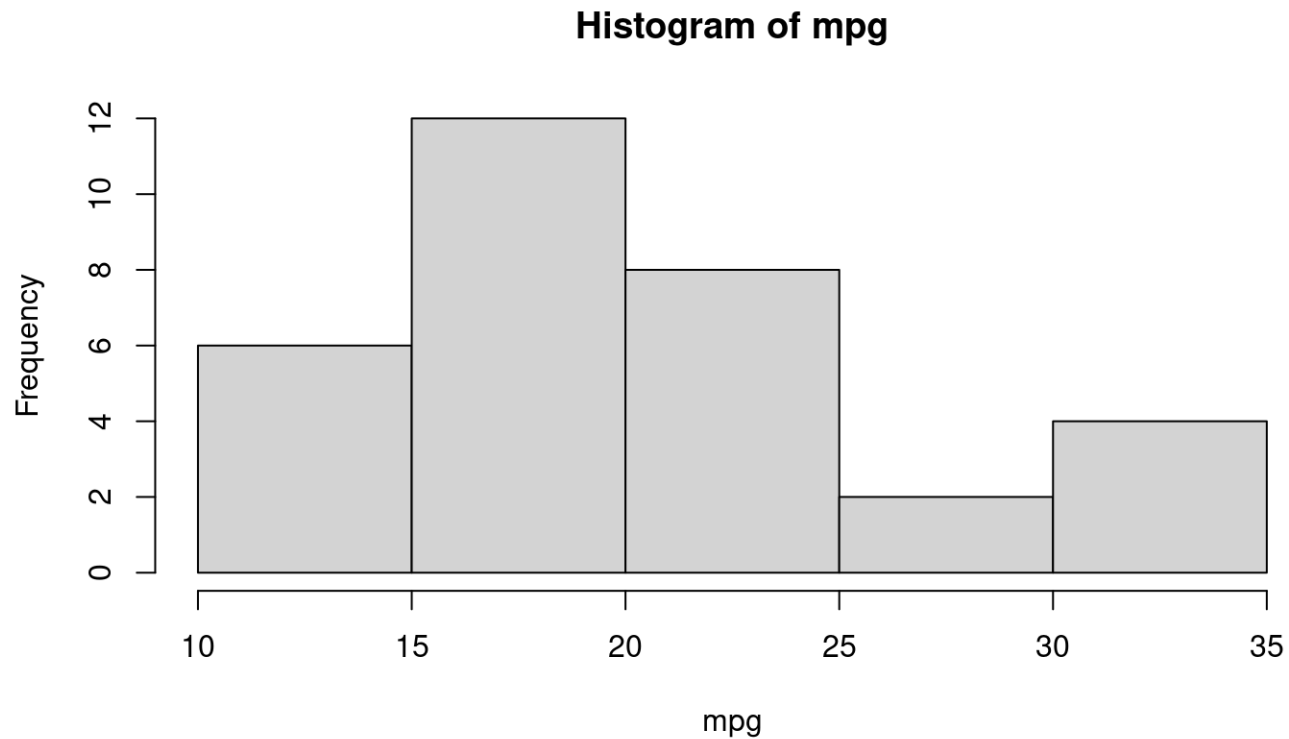
Plotting is an important component of exploratory data analysis. These are some rough plots that you can use in real time while exploring your data.

We will go over formatting and making plots look nicer in additional lectures.

- Basic summarization plots:
 - `hist(x)`: histogram of x
 - `plot(x, y)`: scatterplot of x and y
 - `boxplot(y~x)`: boxplot of y against levels of x

Histogram

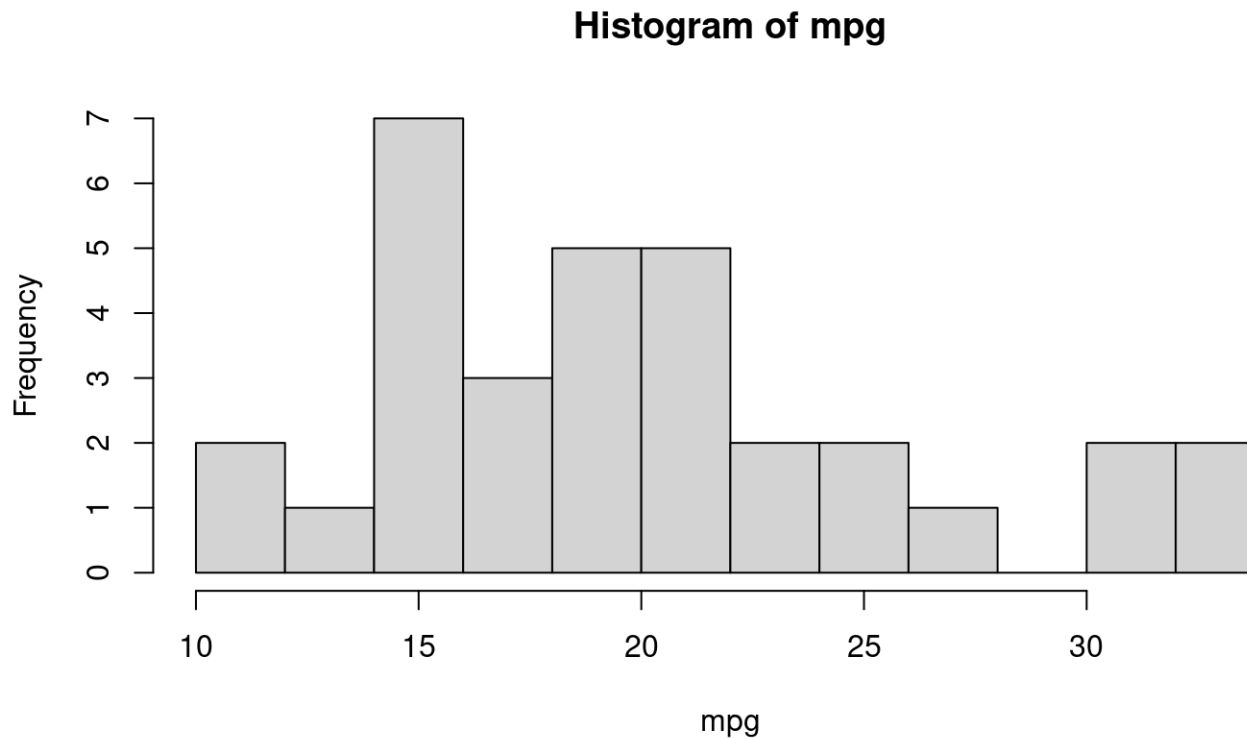
```
mpg <- jhu_cars %>% pull(mpg)  
hist(x = mpg)
```



Histogram

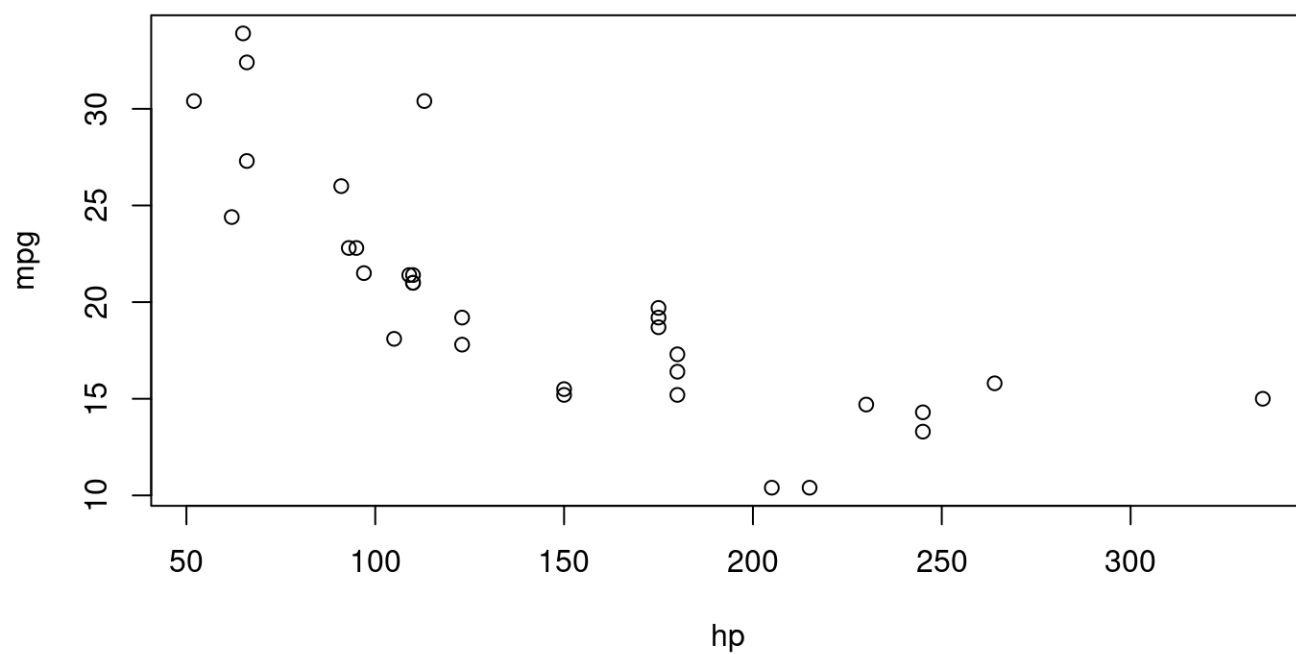
Use the `breaks` = argument to tweak the resolution:

```
hist(x = mpg, breaks = 10)
```



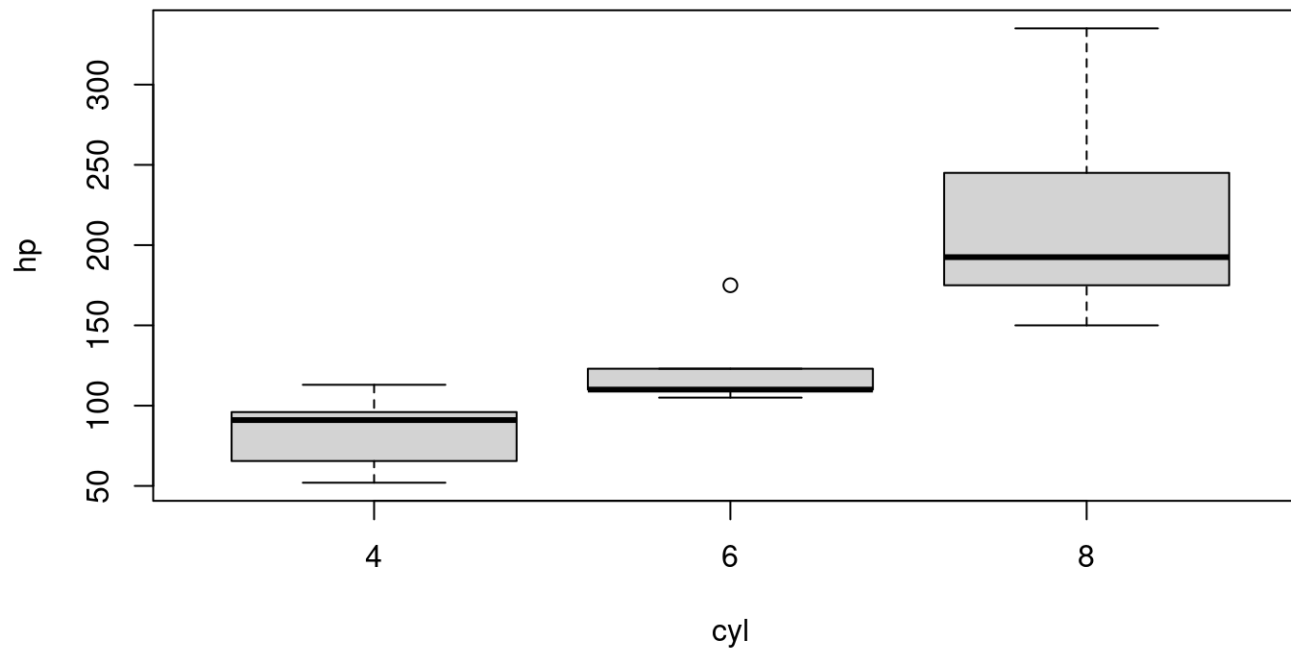
Scatterplot

```
mpg <- jhu_cars %>% pull(mpg)
hp <- jhu_cars %>% pull(hp)
plot(x = hp, y = mpg) # plot(hp, mpg)
```



Boxplot

```
cyl <- jhu_cars %>% pull(cyl)  
hp <- jhu_cars %>% pull(hp)  
boxplot(formula = hp ~ cyl)
```



Summary & Lab Part 3

- `plot(x,y)` and `hist(x)` are great for a **quick snapshot** of the data
- `boxplot(y ~ x)` works for categorical data

[Class Website](#)

[Lab](#)



Image by [Gerd Altmann](#) from [Pixabay](#)