

Data Summarization

Introduction to R for Public Health Researchers

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 quantities of `x`. Default is min, IQR, max
 - `range(x)`: displays the range. Same as `c(min(x), max(x))`
 - `sum(x)`: sum of `x`
 - **all have a** `na.rm` for missing data - discussed later
- Transformations
 - `log` - log (base `e`) transformation
 - `log2` - log base 2 transform
 - `log10` - log base 10 transform
 - `sqrt` - square root

Some examples

We can use the `mtcars` to explore different ways of summarizing data. The `head` command displays the first 6 (default) rows of an object:

```
head(mtcars)
```

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

Statistical summarization

Note - the `$` references/selects columns from a `data.frame/tibble`:

```
mean(mtcars$hp)
```

```
[1] 146.6875
```

```
quantile(mtcars$hp)
```

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

Statistical summarization

```
median(mtcars$wt)
```

```
[1] 3.325
```

```
quantile(mtcars$wt, probs = 0.6)
```

```
60%  
3.44
```

Statistical summarization

`t.test` will be covered more in detail later, gives a mean and 95% CI:

```
t.test(mtcars$wt)
```

One Sample t-test

```
data:  mtcars$wt
t = 18.6, df = 31, p-value < 2.2e-16
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 2.864478 3.570022
sample estimates:
mean of x
 3.21725
```

```
broom::tidy(t.test(mtcars$wt))
```

```
# A tibble: 1 x 8
  estimate statistic  p.value parameter conf.low conf.high method
  <dbl>      <dbl>    <dbl>      <dbl>    <dbl>    <dbl> <chr>
1     3.22      18.6 2.26e-18         31     2.86     3.57 One S...
# ... with 1 more variable: alternative <chr>
```

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, NA, 4, 2, 8, 10, 45, 42)
mean(x)
```

```
[1] NA
```

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

```
[1] 13.77778
```

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

0%	25%	50%	75%	100%
1	4	7	10	45

Data Summarization on matrices/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
- The `matrixStats` package has additional `row*` and `col*` functions
 - Like `rowSds`, `colQuantiles`

Lab Part 1

[Website](#)

TB Incidence

Please download the TB incidence data:

http://johnmuschelli.com/intro_to_r/data/tb_incidence.xlsx

Here we will read in a `tibble` of values from TB incidence:

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

```
[1] "TB incidence, all forms (per 100 000 population per year)"
[2] "1990"
[3] "1991"
[4] "1992"
[5] "1993"
[6] "1994"
[7] "1995"
[8] "1996"
[9] "1997"
[10] "1998"
[11] "1999"
[12] "2000"
[13] "2001"
[14] "2002"
[15] "2003"
[16] "2004"
```

TB Incidence

Reading in the data is useful, but for following along, use:

```
tb = jhur::read_tb()
```

Indicator of TB

We can rename the first column to be the country measured using the `rename` function in `dplyr` (we have to use the ``` things because there are spaces in the name):

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

`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"
```

Column and Row means

`colMeans` and `rowMeans` must work on all numeric data. We will subset years before 2000 (starting with 1):

```
avgs = select(tb, starts_with("1"))
colMeans(avgs, 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						
	121.5169	125.0435						

```
tb$before_2000_avg = rowMeans(avgs, na.rm = TRUE)
head(tb[, c("country", "before_2000_avg")])
```

```
# A tibble: 6 x 2
  country      before_2000_avg
  <chr>          <dbl>
1 Afghanistan    168
2 Albania         26.3
3 Algeria         41.8
4 American Samoa   8.5
5 Andorra         28.8
6 Angola         225.
```

Summarize the data: `dplyr::summarize` function

`dplyr::summarize` will allow you to summarize data. If you would like to summarize **all** columns, you can use `summarize_all` and pass in a function (with other arguments):

```
summarize_all(avgs, mean, na.rm = TRUE)
```

```
# A tibble: 1 x 10
  `1990` `1991` `1992` `1993` `1994` `1995` `1996` `1997` `1998` `1999`
  <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1  106.  108.  108.  110.  112.  114.  115.  119.  122.  125.
```

Summary Function

Using `summary` can give you rough snapshots of each column, but you would likely use `mean`, `min`, `max`, and `quantile` when necessary:

```
summary(tb)
```

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

Youth Tobacco Survey

Please download the Youth Tobacco Survey data.

```
yts = jhur::read_yts()
head(yts)
```

```
# A tibble: 6 x 31
  YEAR LocationAbbr LocationDesc TopicType TopicDesc MeasureDesc
<int> <chr>         <chr>         <chr>      <chr>      <chr>
1  2015 AZ          Arizona      Tobacco ... Cessatio... Percent of...
2  2015 AZ          Arizona      Tobacco ... Cessatio... Percent of...
3  2015 AZ          Arizona      Tobacco ... Cessatio... Percent of...
4  2015 AZ          Arizona      Tobacco ... Cessatio... Quit Attem...
5  2015 AZ          Arizona      Tobacco ... Cessatio... Quit Attem...
6  2015 AZ          Arizona      Tobacco ... Cessatio... Quit Attem...
# ... with 25 more variables: DataSource <chr>, 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 <int>, Gender <chr>,
#   Race <chr>, Age <chr>, Education <chr>, GeoLocation <chr>,
#   TopicTypeId <chr>, TopicId <chr>, MeasureId <chr>,
#   StratificationID1 <chr>, StratificationID2 <chr>,
#   StratificationID3 <chr>, StratificationID4 <chr>, SubMeasureID <chr>,
#   DisplayOrder <int>
```


Youth Tobacco Survey: Reading in yourself

Please download the Youth Tobacco Survey data. You can also read it in directly from the web:

```
library(readr)
yts = read_csv(
  "http://johnmuschelli.com/intro_to_r/data/Youth_Tobacco_Survey_YTS_Data.csv"
)
head(yts)
```

```
# A tibble: 6 x 31
  YEAR LocationAbbr LocationDesc TopicType TopicDesc MeasureDesc
  <int> <chr>         <chr>         <chr>    <chr>    <chr>
1  2015 AZ          Arizona      Tobacco ... Cessatio... Percent of...
2  2015 AZ          Arizona      Tobacco ... Cessatio... Percent of...
3  2015 AZ          Arizona      Tobacco ... Cessatio... Percent of...
4  2015 AZ          Arizona      Tobacco ... Cessatio... Quit Attem...
5  2015 AZ          Arizona      Tobacco ... Cessatio... Quit Attem...
6  2015 AZ          Arizona      Tobacco ... Cessatio... Quit Attem...
# ... with 25 more variables: DataSource <chr>, 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 <int>, Gender <chr>,
#   Race <chr>, Age <chr>, Education <chr>, GeoLocation <chr>,
#   TopicTypeId <chr>, TopicId <chr>, MeasureId <chr>,
#   StratificationID1 <chr>, StratificationID2 <chr>,
#   StratificationID3 <chr>, StratificationID4 <chr>, SubMeasureID <chr>,
#   DisplayOrder <int>
```

Length and unique

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

```
unique(yts$LocationDesc)[1:10]
```

```
[1] "Arizona"           "Connecticut"  
[3] "Georgia"           "Hawaii"  
[5] "Illinois"          "Louisiana"  
[7] "Mississippi"       "Utah"  
[9] "Missouri"          "National (States and DC)"
```

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

```
length(unique(yts$LocationDesc))
```

```
[1] 50
```

Table

`table(x)` will return a frequency table of unique elements of `x`

```
table(yts$LocationDesc)[1:5]
```

Alabama	Arizona	Arkansas	California	Colorado
378	240	210	96	48

Lab Part 2

[Website](#)

dplyr: count

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

```
# A tibble: 50 x 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
```

dplyr: count

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

```
# A tibble: 50 x 3
  LocationDesc      Age      n
  <chr>          <chr> <int>
1 Alabama      All Ages   378
2 Arizona      All Ages   240
3 Arkansas     All Ages   210
4 California   All Ages    96
5 Colorado     All Ages    48
6 Connecticut  All Ages   384
7 Delaware     All Ages   312
8 District of Columbia All Ages    48
9 Florida      All Ages    96
10 Georgia     All Ages   282
# ... with 40 more rows
```

Subsetting to specific columns

Let's just take smoking status measures for all genders in middle school using `filter`, and the columns that represent the year, state using `select`:

```
library(dplyr)
sub_yts = filter(yts, MeasureDesc == "Smoking Status",
                  Gender == "Overall", Response == "Current",
                  Education == "Middle School")
sub_yts = select(sub_yts, YEAR, LocationDesc, Data_Value)
head(sub_yts, 4)
```

```
# A tibble: 4 x 3
  YEAR LocationDesc Data_Value
<int> <chr>         <dbl>
1  2015 Arizona         3.2
2  2015 Connecticut     0.8
3  2015 Hawaii          3
4  2015 Illinois        2
```

Perform Operations By Groups: dplyr

`group_by` allows you group the data set by grouping variables:

```
sub_yts = group_by(sub_yts, YEAR)
head(sub_yts)
```

```
# A tibble: 6 x 3
# Groups:   YEAR [1]
  YEAR LocationDesc Data_Value
  <int> <chr>         <dbl>
1  2015 Arizona          3.2
2  2015 Connecticut      0.8
3  2015 Hawaii           3
4  2015 Illinois         2
5  2015 Louisiana        5.2
6  2015 Mississippi     4.7
```

- doesn't change the data in any way, but how **functions operate on it**

Summarize the data

```
summarize(sub_yts, year_avg = mean(Data_Value, na.rm = TRUE))
```

```
# A tibble: 17 x 2
```

	YEAR	year_avg
	<int>	<dbl>
1	1999	14.6
2	2000	12.5
3	2001	9.84
4	2002	9.60
5	2003	7.49
6	2004	8.2
7	2005	7.27
8	2006	7.37
9	2007	6.68
10	2008	5.95
11	2009	5.84
12	2010	5.6
13	2011	5.15
14	2012	4.72
15	2013	3.76
16	2014	2.93
17	2015	2.86

Ungroup the data

You usually want to perform operations on groups and may want to redefine the groups. The `ungroup` function will allow you to clear the groups from the data:

```
sub_yts = ungroup(sub_yts)
sub_yts
```

```
# A tibble: 222 x 3
  YEAR LocationDesc Data_Value
  <int> <chr>         <dbl>
1  2015 Arizona          3.2
2  2015 Connecticut      0.8
3  2015 Hawaii           3
4  2015 Illinois         2
5  2015 Louisiana       5.2
6  2015 Mississippi     4.7
7  2015 Missouri        2.4
8  2015 New Jersey      1.2
9  2015 North Carolina  2.3
10 2015 North Dakota    3.6
# ... with 212 more rows
```

Using the pipe

Pipe `sub_yts` into `group_by`, then pipe that into `summarize`:

```
yts_avgs = sub_yts %>%  
  group_by(YEAR) %>%  
  summarize(year_avg = mean(Data_Value, na.rm = TRUE))  
head(yts_avgs)
```

```
# A tibble: 6 x 2  
  YEAR year_avg  
  <int>   <dbl>  
1  1999    14.6  
2  2000    12.5  
3  2001     9.84  
4  2002     9.60  
5  2003     7.49  
6  2004     8.2
```

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:

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

```
# A tibble: 222 x 4  
# Groups:   YEAR [17]  
   YEAR LocationDesc Data_Value year_avg  
   <int> <chr>         <dbl>    <dbl>  
1  2000 Alabama      19.1     12.5  
2  2002 Alabama      15.6     9.60  
3  2004 Alabama      13.1     8.2  
4  2006 Alabama       13     7.37  
5  2008 Alabama       8.7     5.95  
6  2010 Alabama       7      5.6  
7  2012 Alabama       7.5     4.72  
8  2014 Alabama       6.4     2.93  
9  2000 Arizona      11.4     12.5  
10 2003 Arizona       8.7     7.49  
# ... with 212 more rows
```

Counting

Standard statistics can be calculated. There are other functions, such as `n()` count the number of observations, `tally()` to count as a wrapper:

```
sub_yts %>%  
  group_by(YEAR) %>%  
  summarize(n = n()) %>%  
  head
```

```
# A tibble: 6 x 2  
  YEAR      n  
  <int> <int>  
1  1999     10  
2  2000     27  
3  2001     11  
4  2002     23  
5  2003     12  
6  2004     14
```

```
sub_yts %>%  
  group_by(YEAR) %>%  
  tally() %>%  
  head
```

```
# A tibble: 6 x 2  
  YEAR      n  
  <int> <int>  
1  1999     10  
2  2000     27  
3  2001     11  
4  2002     23  
5  2003     12  
6  2004     14
```

Lab Part 3

[Website](#)

Data Summarization/Visualization

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

Data Summarization/Visualization: ggplot2

ggplot2 is a package of plotting that is very popular and powerful (using the **g**rammar of **g**raphics). We will use `qplot` ("quick plot") for most of the basic examples:

```
qplot
```

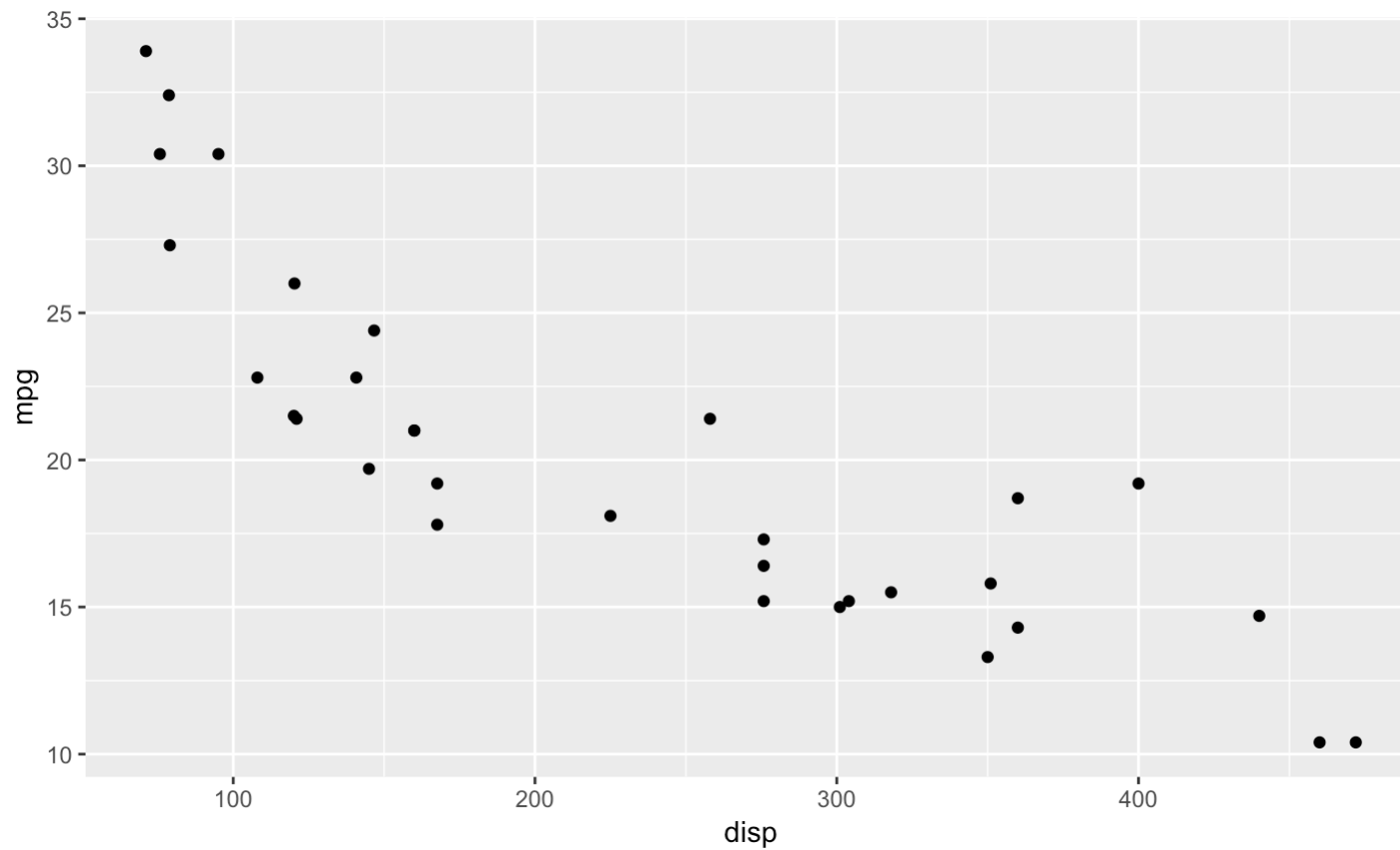
```
function (x, y, ..., data, facets = NULL, margins = FALSE, geom = "auto",  
          xlim = c(NA, NA), ylim = c(NA, NA), log = "", main = NULL,  
          xlab = NULL, ylab = NULL, asp = NA, stat = NULL, position = NULL)  
NULL
```


Basic Plots

Plotting is an important component of exploratory data analysis. We will review some of the more useful and informative plots here. We will go over formatting and making plots look nicer in additional lectures.

Scatterplot

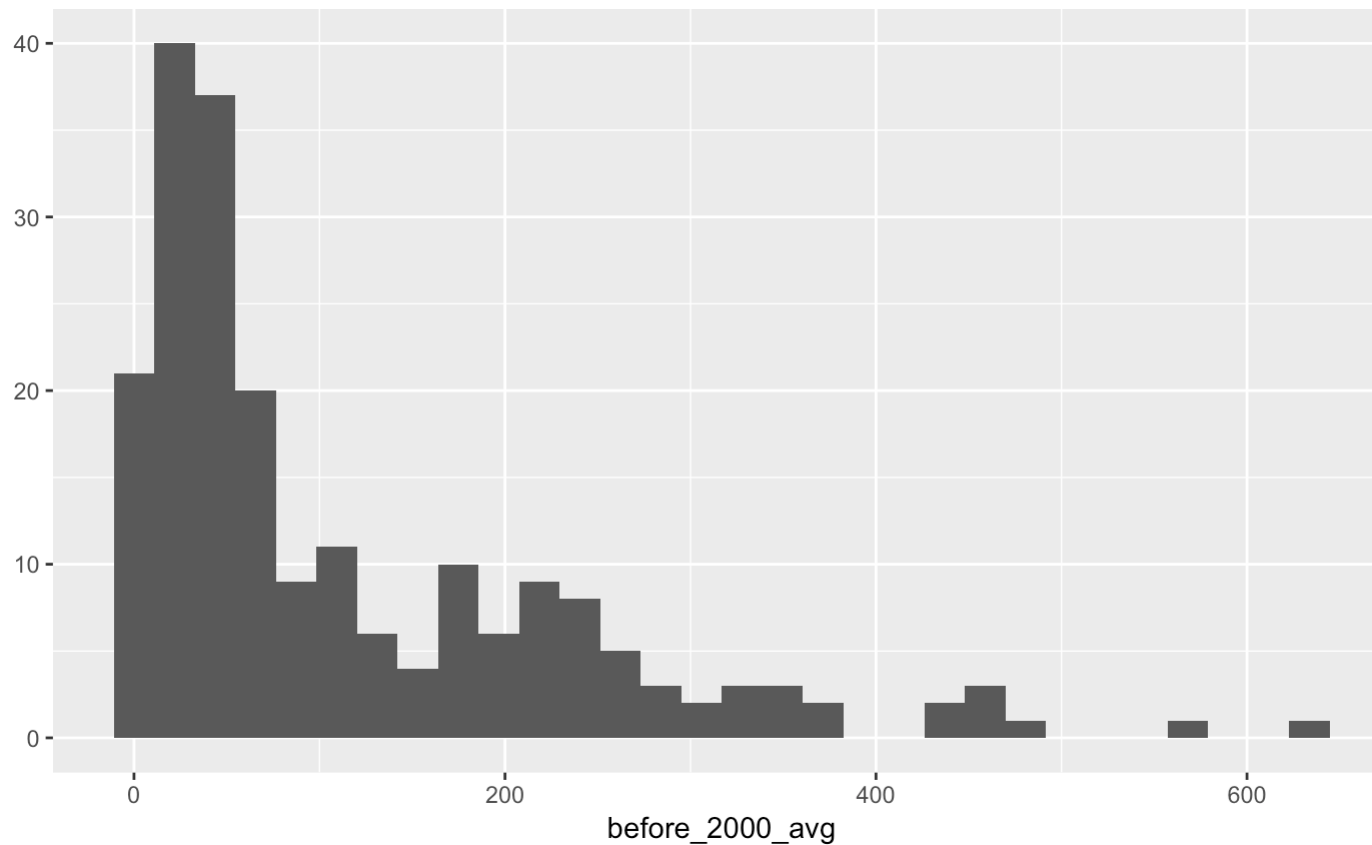
```
library(ggplot2)
qplot(x = disp, y = mpg, data = mtcars)
```



Histograms

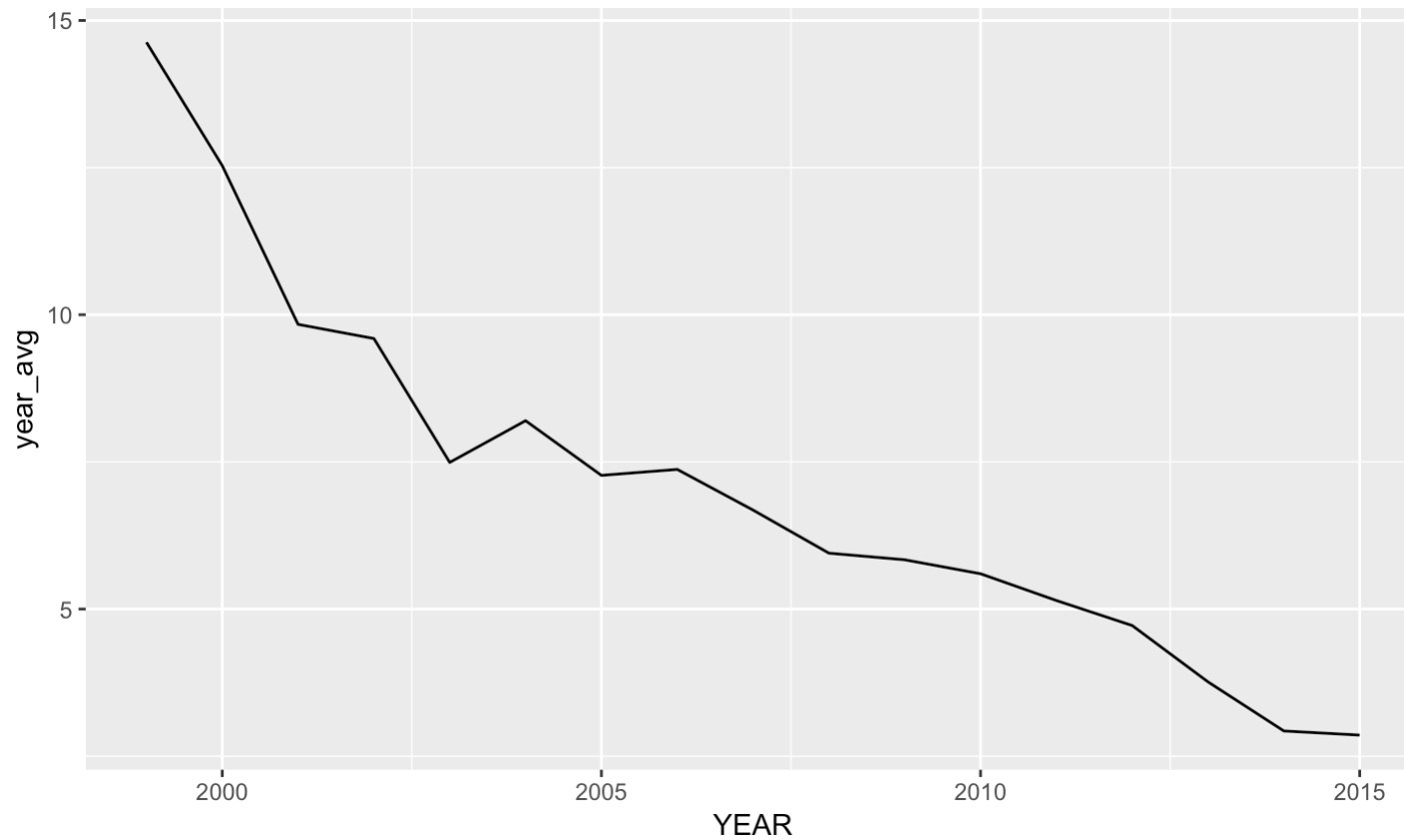
```
qplot(x = before_2000_avg, data = tb, geom = "histogram")
```

Warning: Removed 1 rows containing non-finite values (stat_bin).



Plot with a line

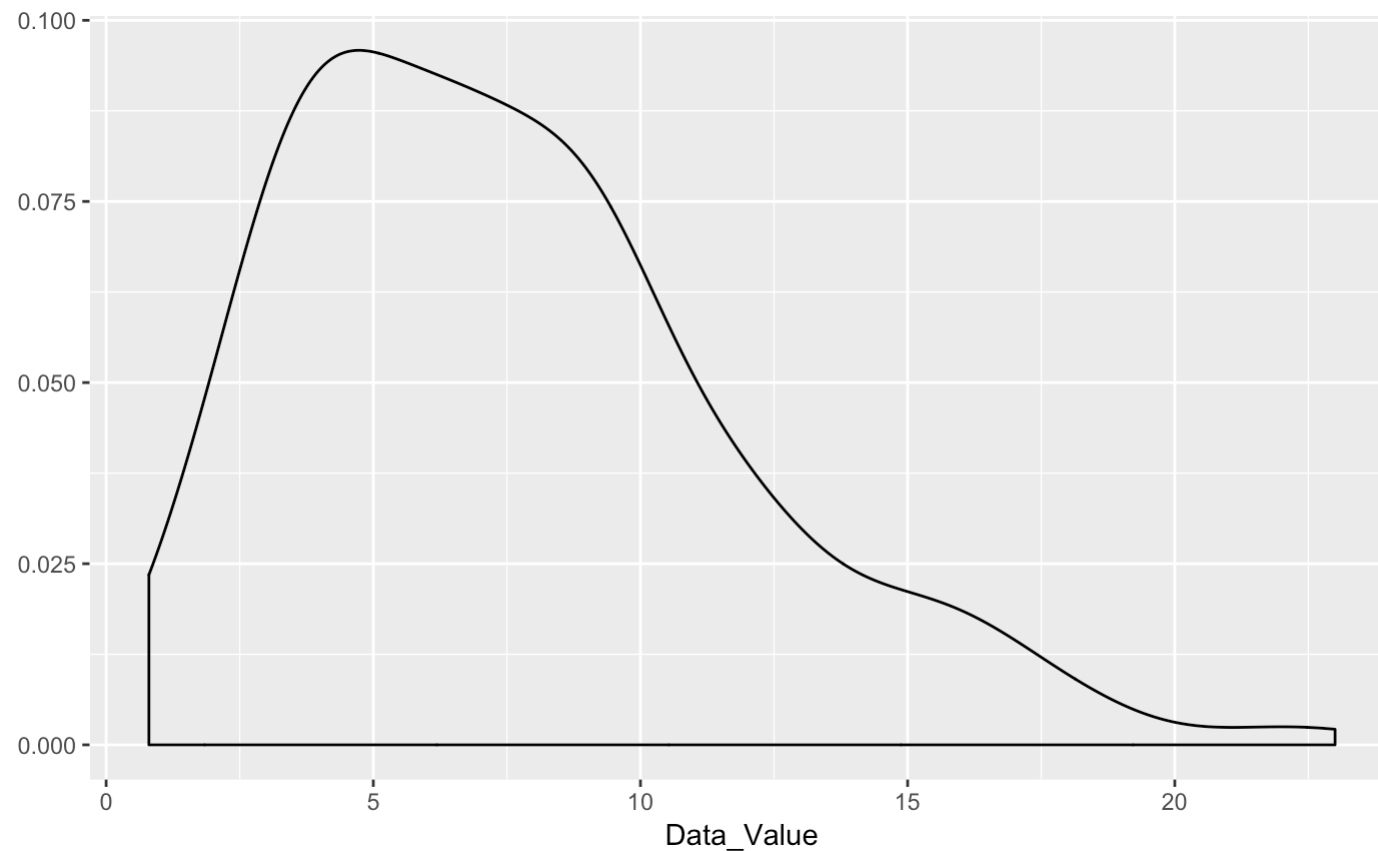
```
qplot(x = YEAR, y = year_avg, data = yts_avgs, geom = "line")
```



Density

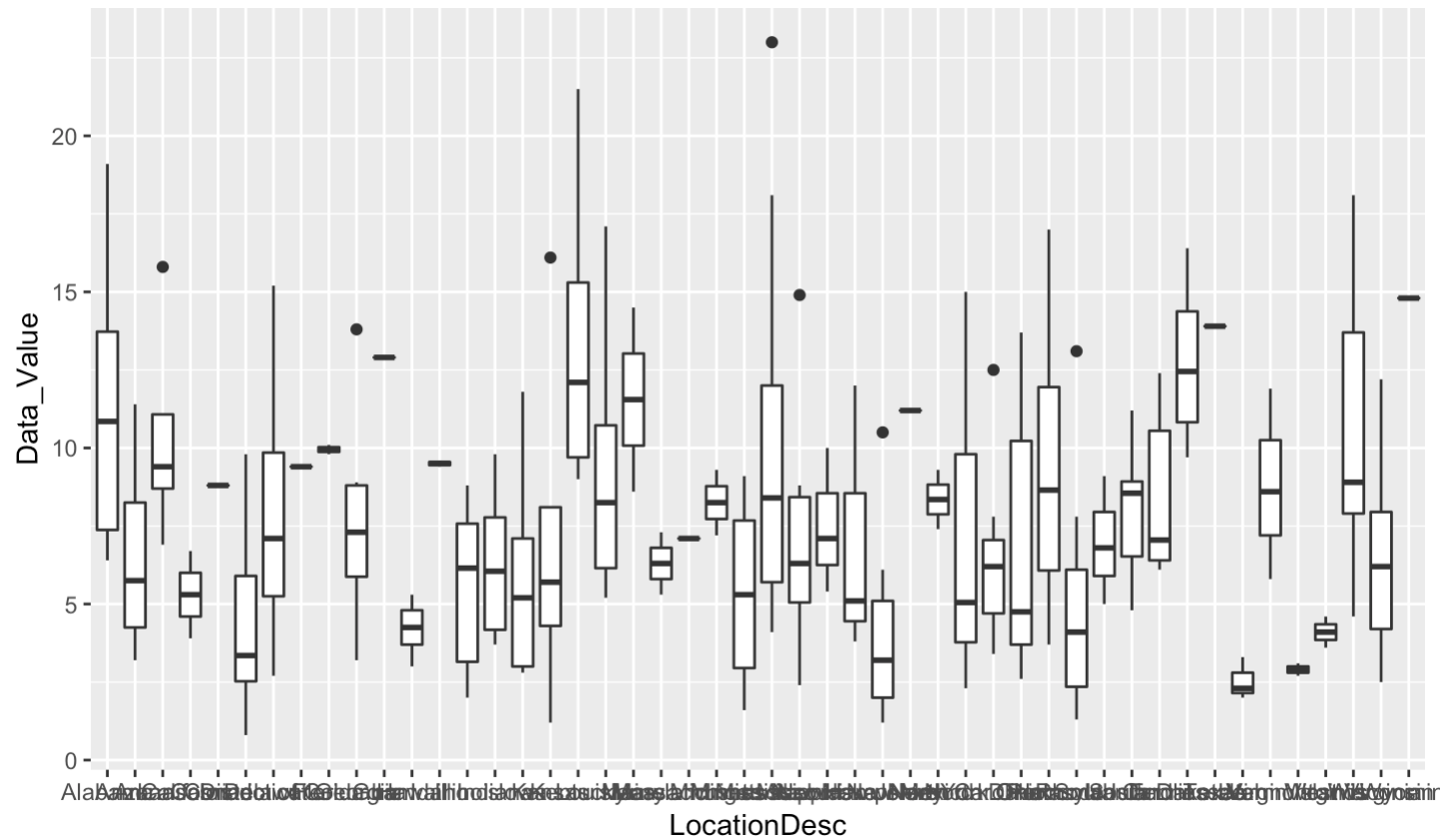
Over all years and states, this is the density of smoking status incidence:

```
qplot(x = Data_Value, data = sub_yts, geom = "density")
```



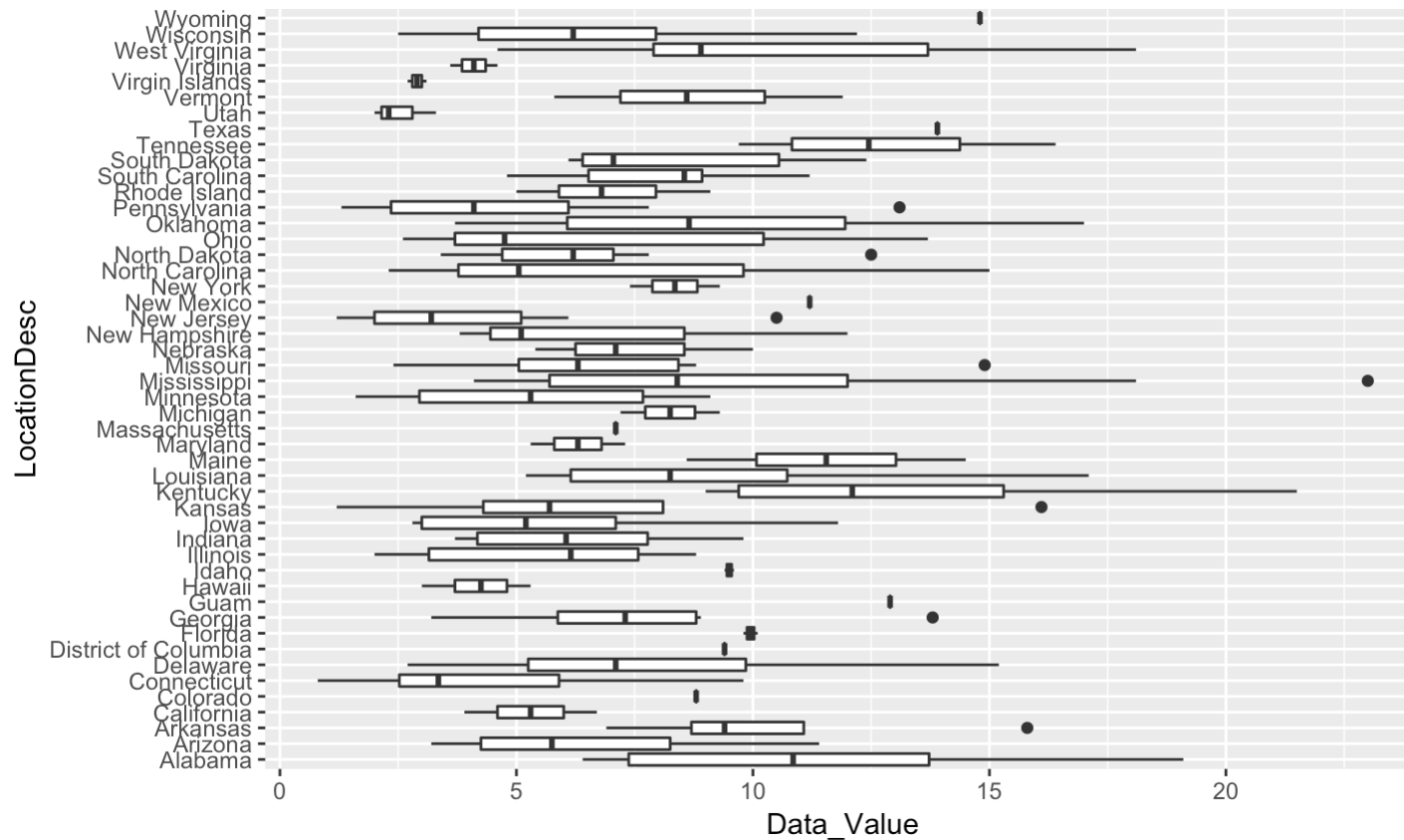
Boxplots

```
qplot(x = LocationDesc, y = Data_Value, data = sub_yts, geom = "boxplot")
```



Boxplots

```
ggplot(x = LocationDesc, y = Data_Value,  
      data = sub_yts, geom = "boxplot") + coord_flip()
```

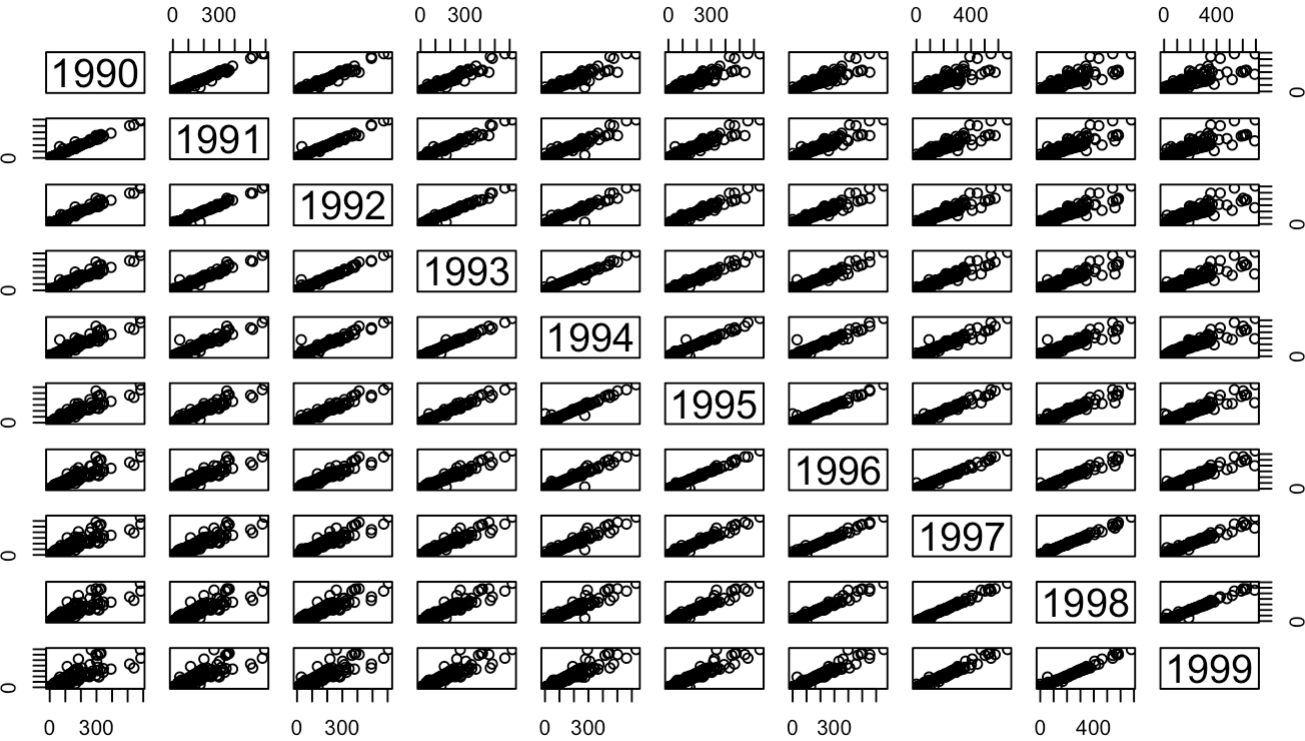


Data Summarization for data.frames

- Basic summarization plots
 - `matplot(x, y)`: scatterplot of two matrices, x and y
 - `pairs(x, y)`: plots pairwise scatter plots of matrices x and y, column by column

Matrix plot

`pairs (avgs)`



Lab Part 4

[Website](#)

Conclusion

- Base R has apply statements that perform things repeatedly.
- `dplyr` has a lot of more intuitive syntax.
 - `group_by` is very powerful, especilly with `summarise/summarize`
- Base R has good things for quickly summarizing rows or colummns of all numeric data.
 - The `matrixStats` package extends this to `colMedians`, `colMaxs`, etc.

Website

[Website](#)

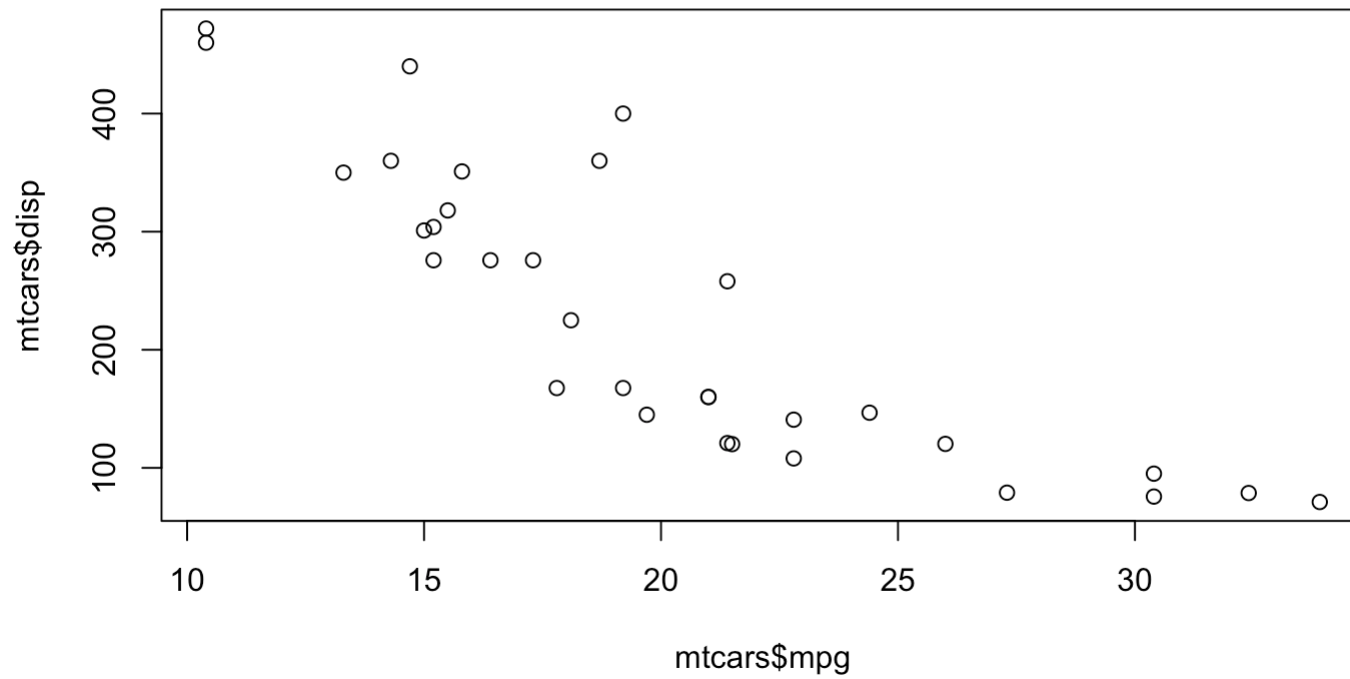
Base R Plots - not covered

Basic Plots

Plotting is an important component of exploratory data analysis. We will review some of the more useful and informative plots here. We will go over formatting and making plots look nicer in additional lectures.

Scatterplot

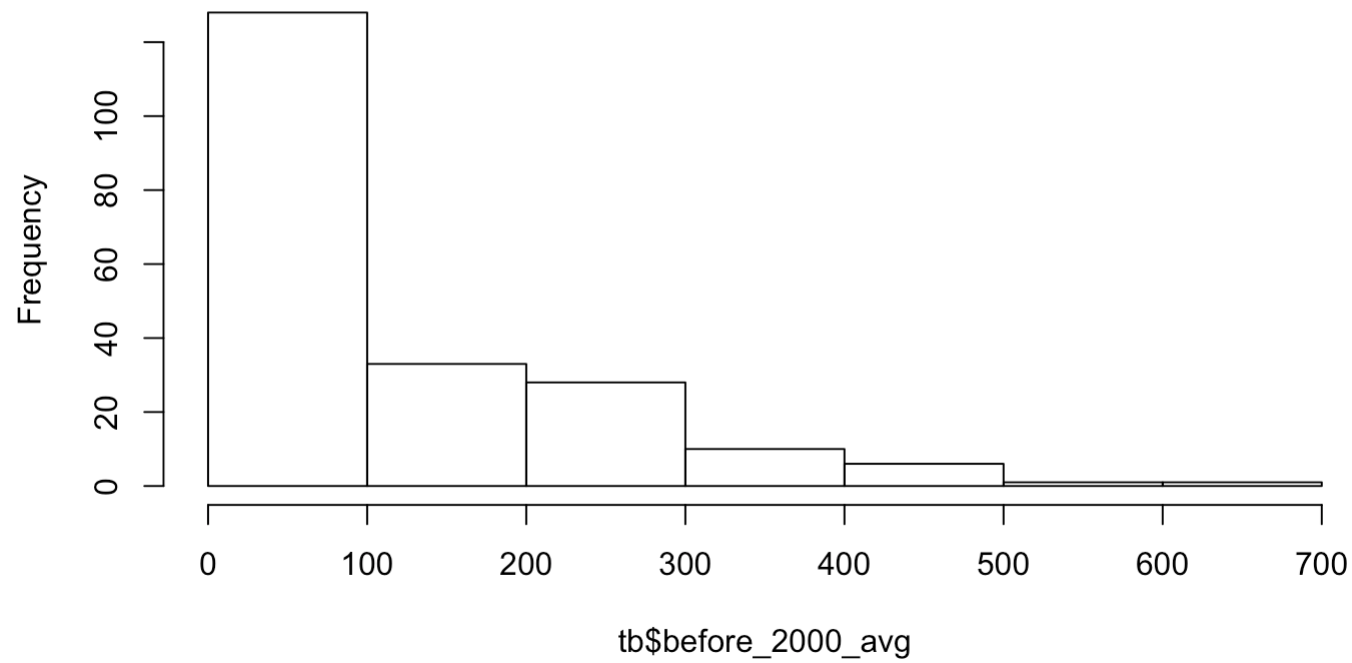
```
plot(mtcars$mpg, mtcars$disp)
```



Histograms

```
hist(tb$before_2000_avg)
```

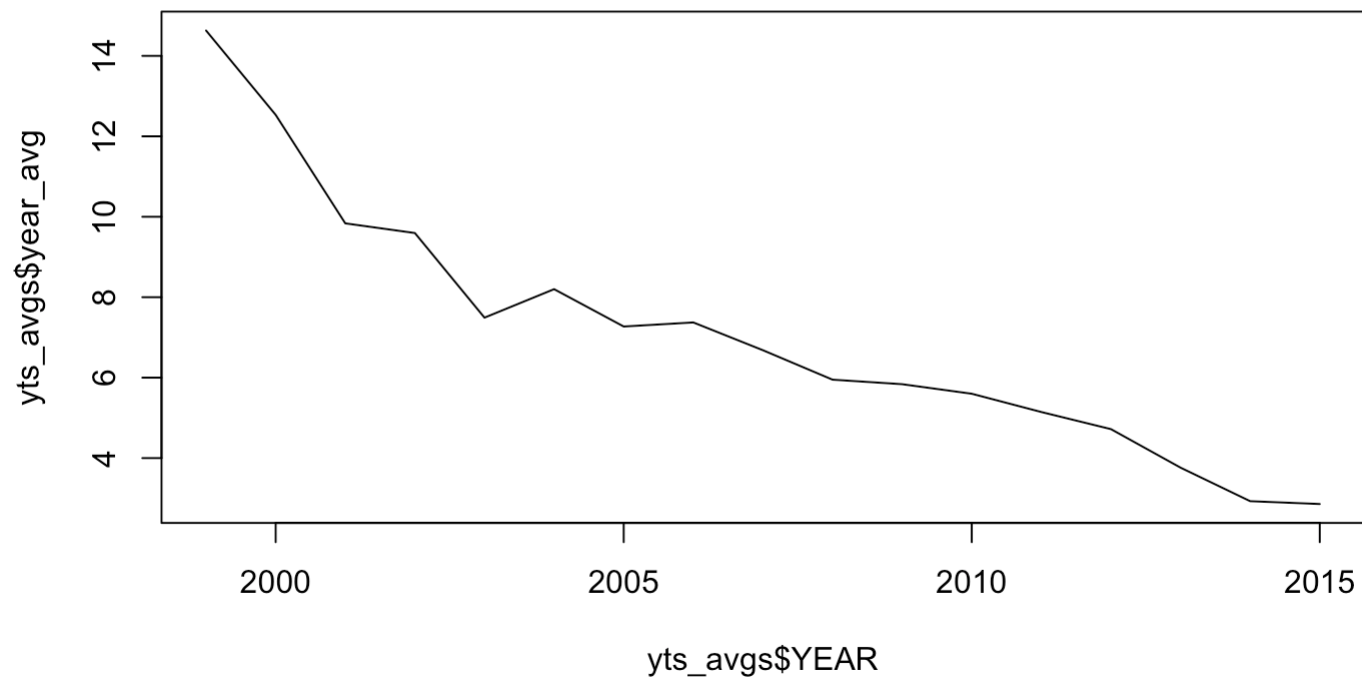
Histogram of tb\$before_2000_avg



Plot with a line

`type = "l"` means a line

```
plot(yts_avgs$YEAR, yts_avgs$year_avg, type = "l")
```

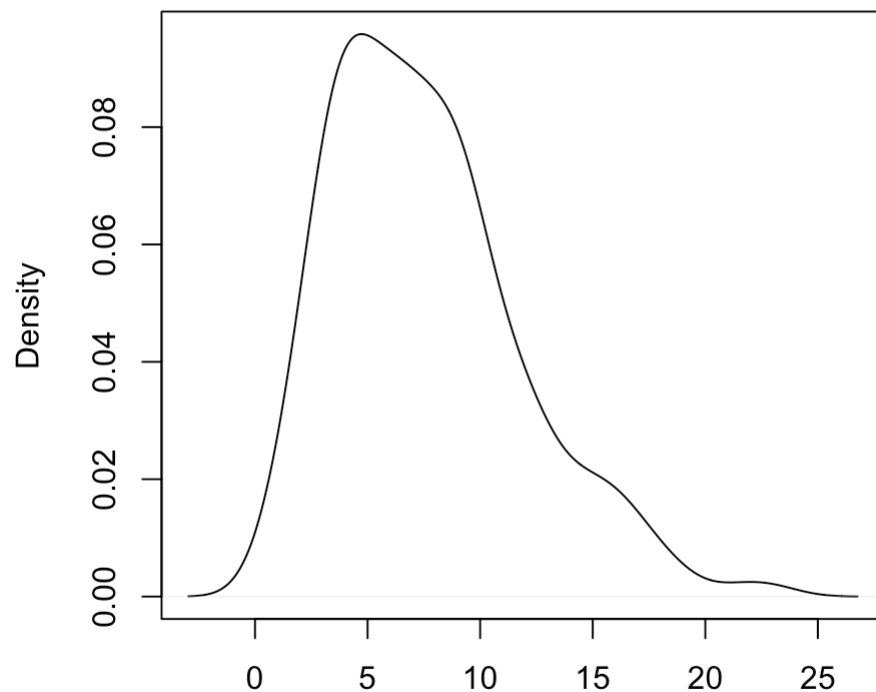


Density

Over all years and states, this is the density of smoking status incidence:

```
plot(density(sub_yts$Data_Value))
```

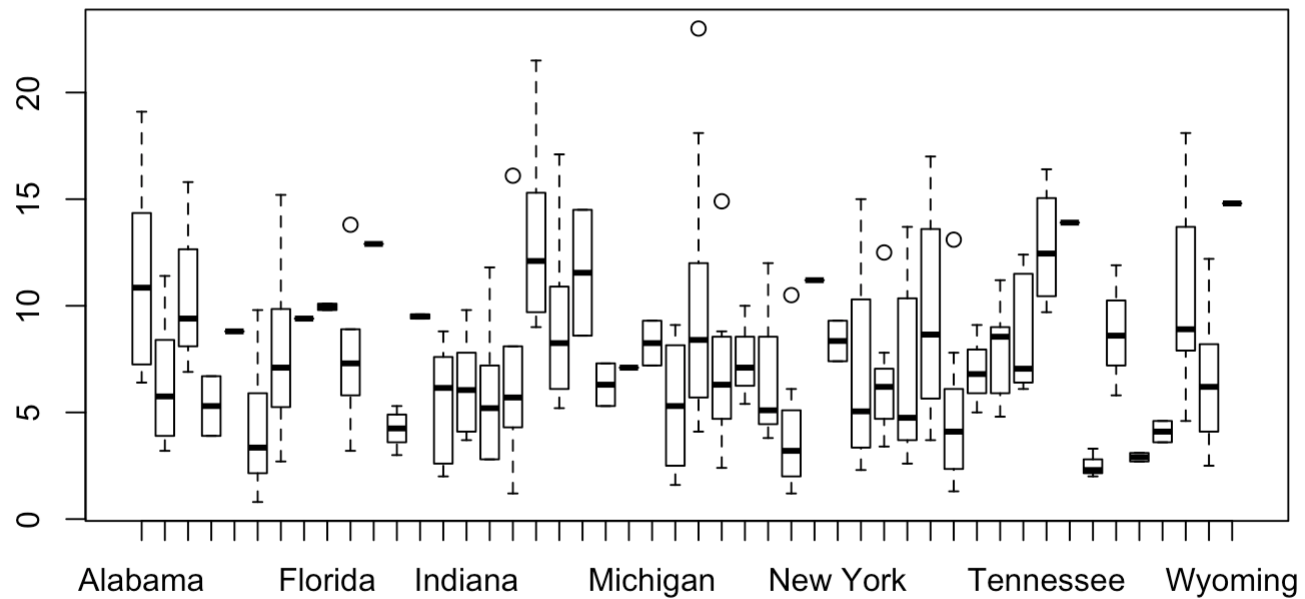
density.default(x = sub_yts\$Data_Value)



N = 222 Bandwidth = 1.254

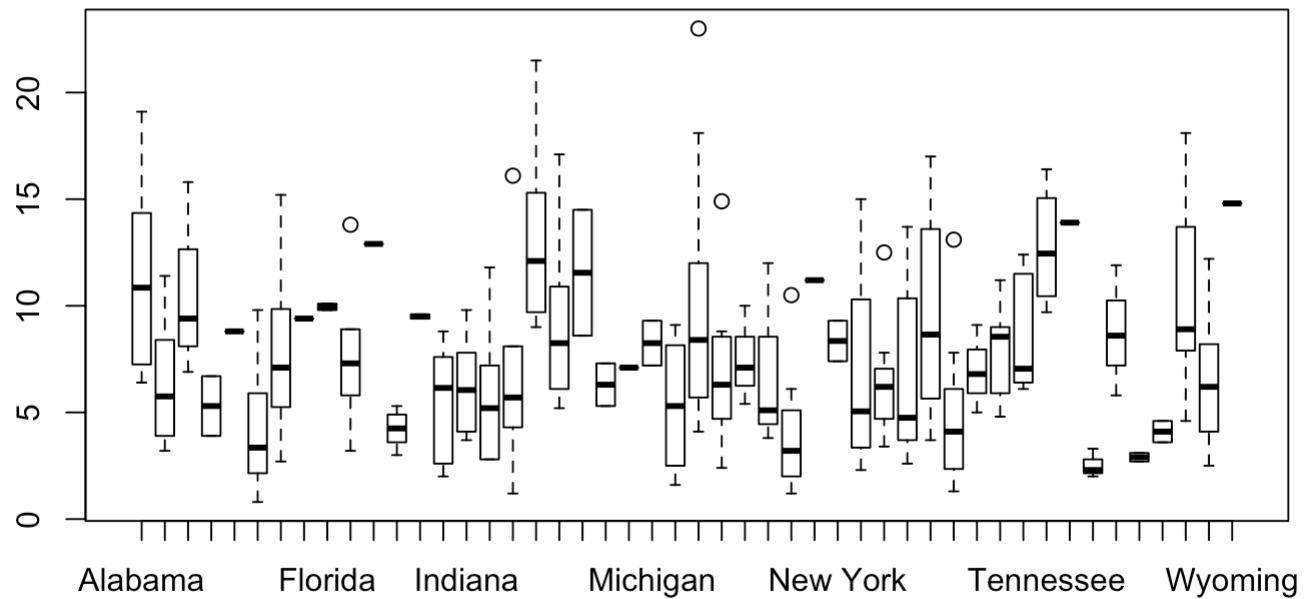
Boxplots

```
boxplot(sub_yts$Data_Value ~ sub_yts$LocationDesc)
```



Boxplots

```
boxplot(Data_Value ~ LocationDesc, data = sub_yts)
```

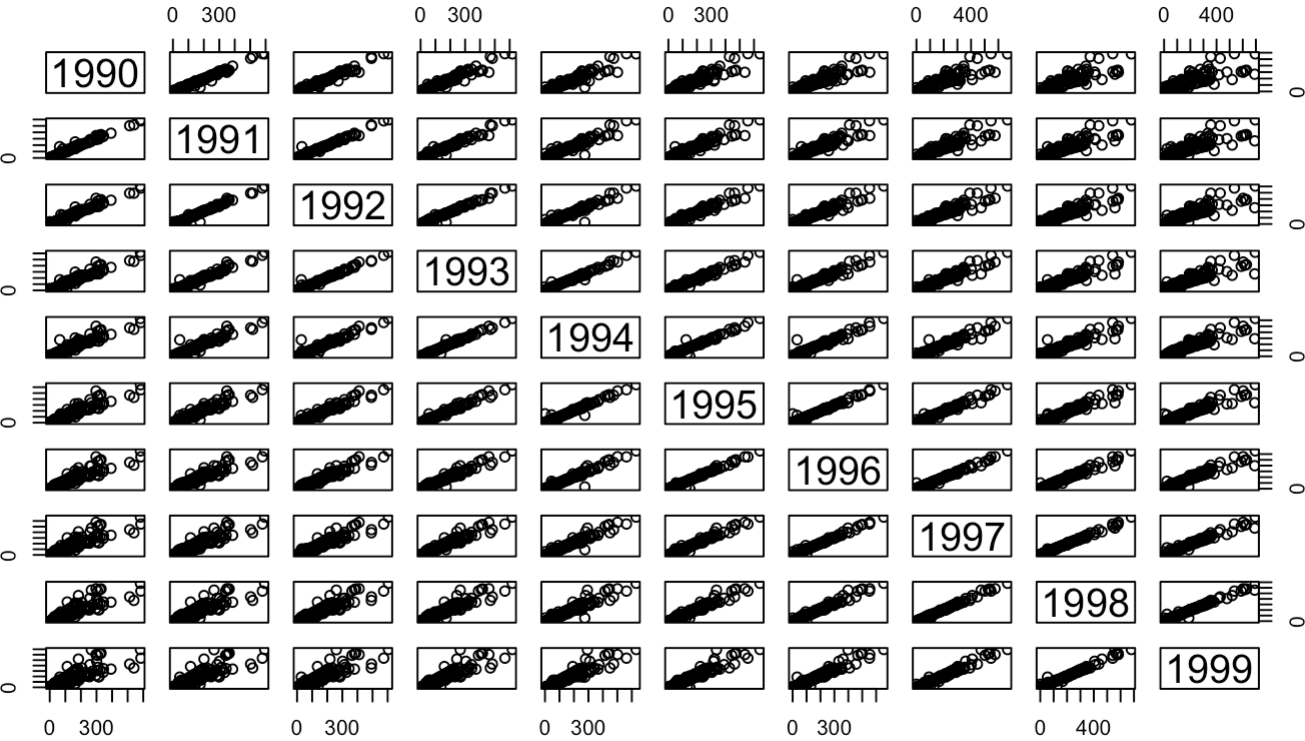


Data Summarization for data.frames

- Basic summarization plots
 - `matplot(x, y)`: scatterplot of two matrices, x and y
 - `pairs(x, y)`: plots pairwise scatter plots of matrices x and y, column by column

Matrix plot

`pairs (avgs)`



Lab Part 4

[Website](#)

Apply statements

You can apply more general functions to the rows or columns of a matrix or data frame, beyond the mean and sum.

```
apply(X, MARGIN, FUN, ...)
```

X : an array, including a matrix.

MARGIN : a vector giving the subscripts which the function will be applied over. E.g., for a matrix 1 indicates rows, 2 indicates columns, c(1, 2) indicates rows and columns. Where X has named dimnames, it can be a character vector selecting dimension names.

FUN : the function to be applied: see 'Details'.

... : optional arguments to FUN.

Apply statements

```
apply(avgs,2,mean, na.rm=TRUE) # column means
```

	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						
	121.5169	125.0435						

```
head(apply(avgs,1,mean, na.rm=TRUE)) # row means
```

```
[1] 168.0 26.3 41.8 8.5 28.8 224.6
```

```
apply(avgs,2,sd, na.rm=TRUE) # columns sds
```

	1990	1991	1992	1993	1994	1995	1996	1997
	110.6440	112.7687	114.4853	116.6744	120.0931	122.7119	126.1800	131.0858
	1998	1999						
	137.3754	146.0755						

```
apply(avgs,2,max, na.rm=TRUE) # column maxs
```

1990	1991	1992	1993	1994	1995	1996	1997	1998	1999
585	594	606	618	630	642	655	668	681	695

Other Apply Statements

- `tapply()`: 'grouping' apply
- `lapply()`: 'list' apply [tomorrow]
- `sapply()`: 'simple' apply [tomorrow]
- Other less used ones...

See more details here: <http://nsaunders.wordpress.com/2010/08/20/a-brief-introduction-to-apply-in-r/>

- Commonly used, but we will discuss how to do all steps in `dplyr`