Big Data Analytics

Organizing and Re-Structuring Data in R

Oftentimes the data you receive is not formatted for visualization, or needs to be transformed. In such cases, R has a variety of function that can help.

Mapping to a Different Scale

- let's say we're interested in visualizing the rent data on a scale of 1-10. If our rent data is not already on this scale, we can easily map it to this scale in R.
- To check the current scale of the rent data, run the following code to find the current maximum and minimum, or simply range of the data.

Find Maximum, Minimum, Range

```
max(census$rent) #find maximum value of rent
min(census$rent) #find minimum value of rent
range(census$rent) #find range of rent
```

Formula to re-scale variable

- **# (new_max - new_min) * ([value] - lowest_value) / (highest_value - # lowest_value) + new_min**
- `(10 - 1) * ((census$rent - min(census$rent))/(max(census$rent) - min(census$rent))) + 1`

Create and Add new variable

- ```
  # it is easy to add a new column: census$newcol <- [formula]
  ```
- ```
  census$rent_10 <- (10 - 1) * ((census$rent - min(census$rent))/(max(census$rent) - min(census$rent))) + 1
  ```

Selecting Subsets

- Another very useful data re-structuring technique involves extracting a subset of existing data. Try the following to subset the census data into 3 different unique data sets, each containing information about one 'region':

```
upper <- census[census$region == "upper", ]
View(upper)
central <- census[census$region == "central", ]
View(central) lower <- census[census$region == "south", ]
View(lower)
```

Scaling and Normalizing
1- Transform and Add data

```
census$population_transf <- (census$population)^5
census$population_transf <- exp(census$population)
census$population_transf <- cos(census$population)
census$population_transf <- abs(census$population)
census$population_transf <- (census$population) * 10
```

Scaling and Normalizing
2- *Log transformations*

- `log(census$population)` `#Computs log base e of population`
- `log(census$population, 2)` `# Computs log base 2 of population`
- `log(census$population, 10)` `# Computs log base 10 of population`

Create new variable from old variables

```
# create a variable of populations density by dividing population size by area
census$pop_density <- census$population/census$area
# log transform your new variable
log(census$pop_density)
```

Aggregating

- `sum(census$population)`
- `mean(census$population)`
- `var(census$population)  #variance`
- `sd(census$population)  # standard deviation`
- `median(log(census$population))`

Data Visualization
- Visualizing 1 Variable

Histograms

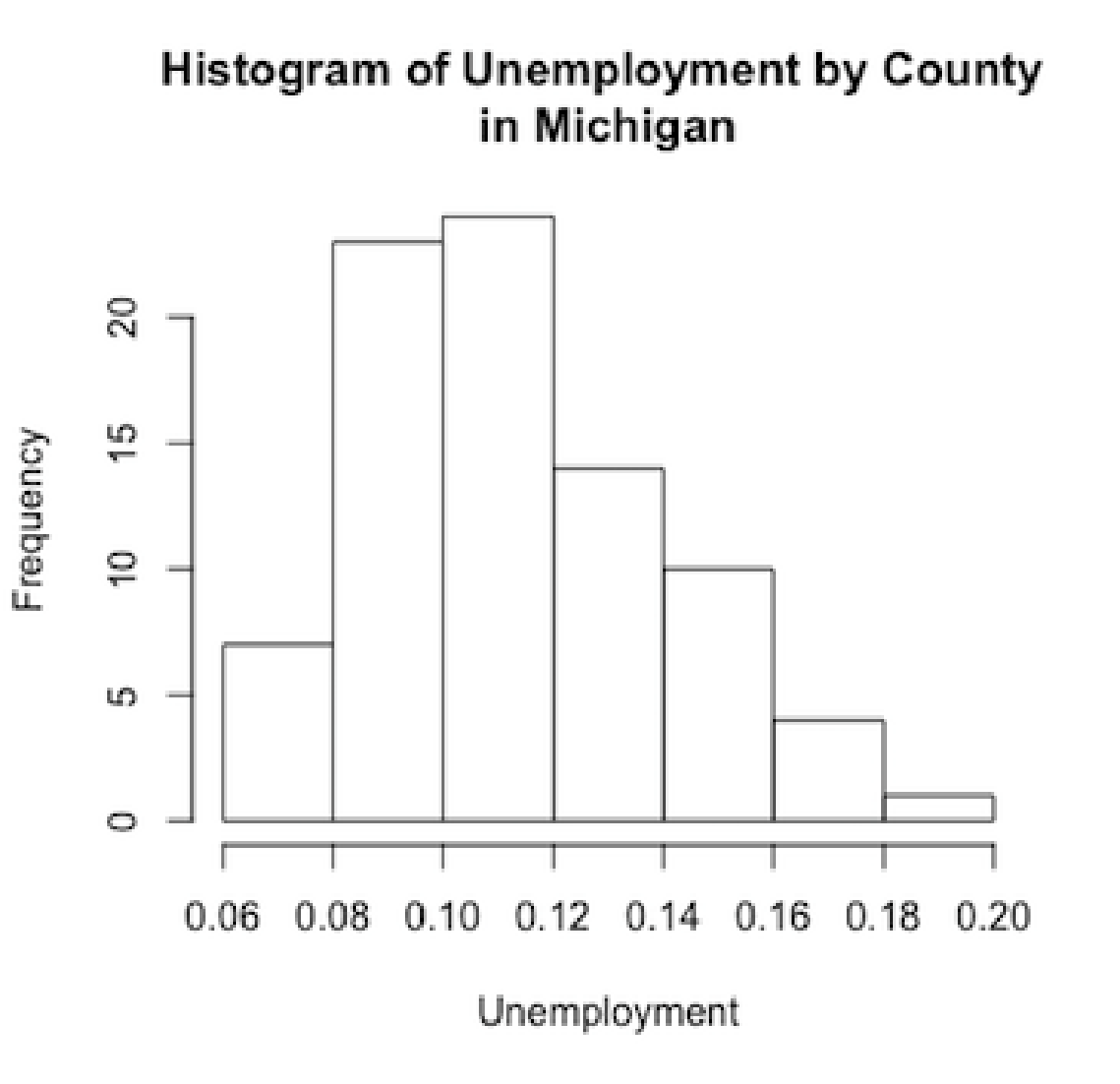
Histogram plots are produced with the **hist()** function in R,
and can be made more readable by adding additional parameters within the function.

- `hist(census$unemploy)  # default graph, without labels`

`# inserting better x-axis label (xlab parameter) and title (main parameter)`

- `hist(census$unemploy, xlab = "Unemployment", main = "Histogram of Unemployment by County \nin Michigan")`

Histograms


Histogram of Unemployment by County in Michigan
Frequency
0
5
10
15
20
0.06
0.08
0.10
0.12
0.14
0.16
0.18
0.20
Unemployment

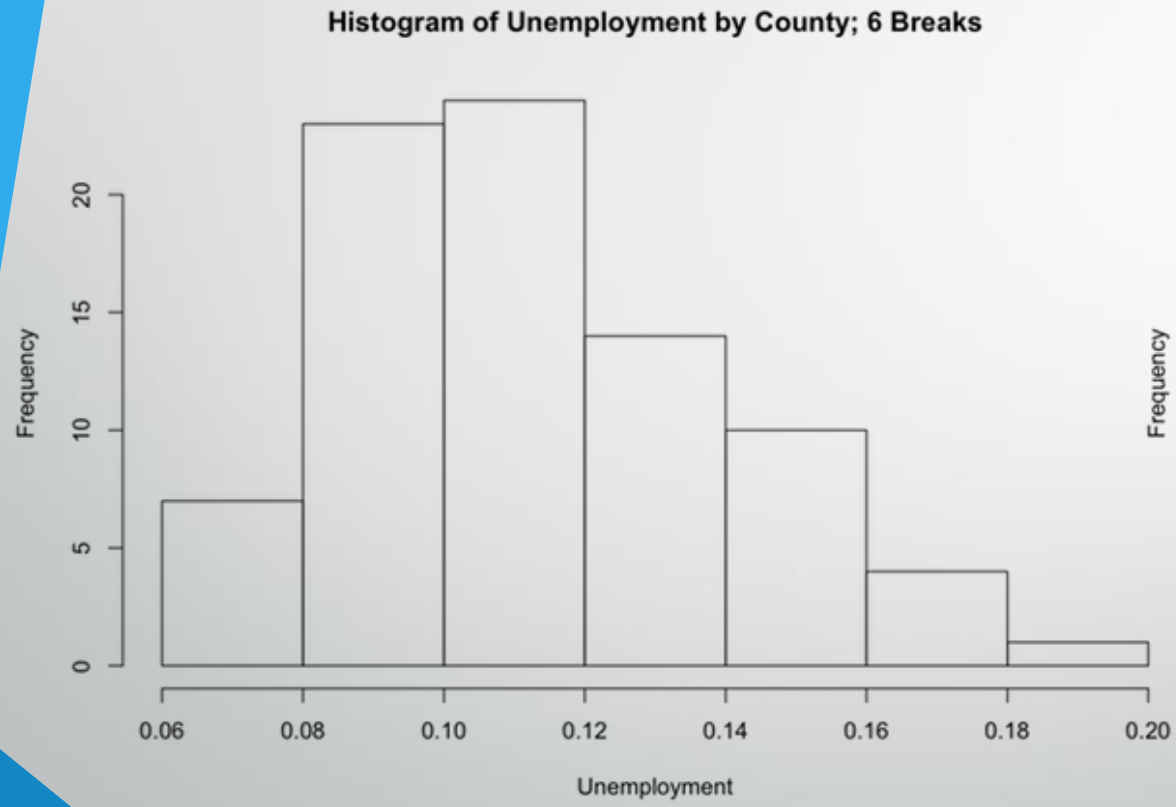
Data Visualization
- Visualizing 1 Variable

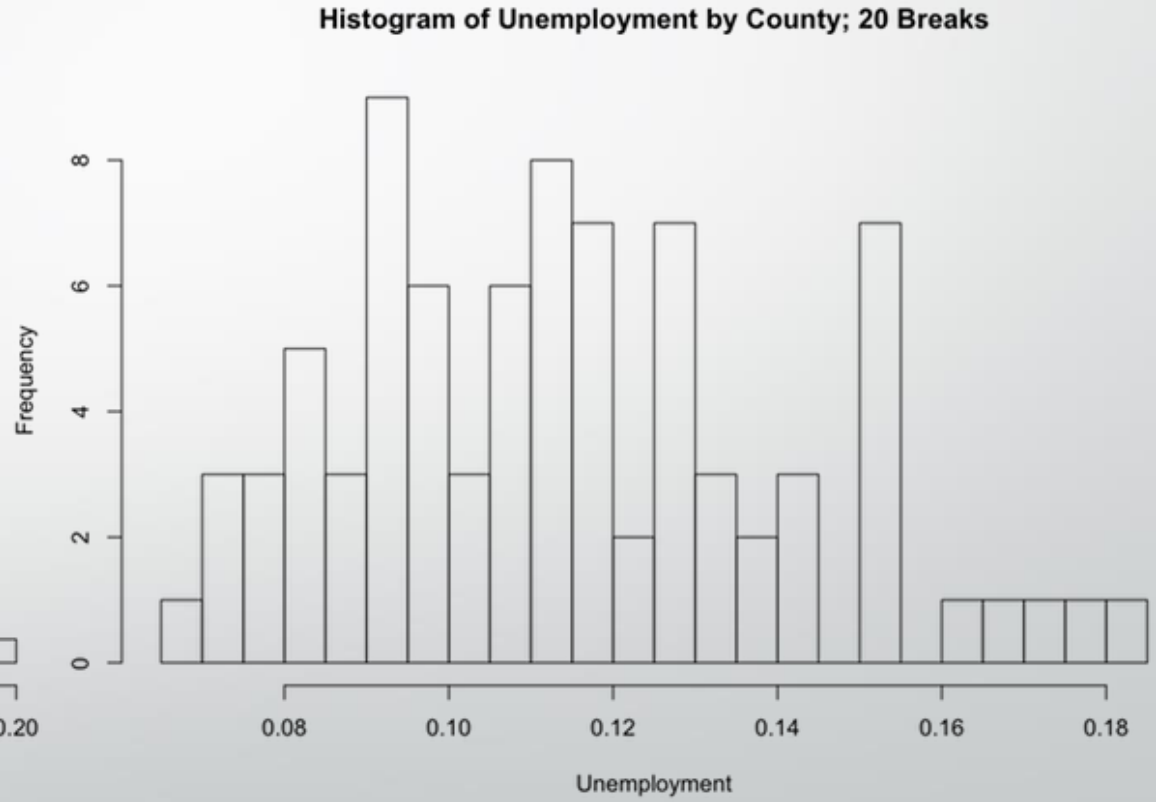
Histogram breaks

An additional useful parameter is breaks, which allow users to increase or decrease the bin-size of the histogram. Notice how the choice of breaks alters the visualization:

- ```
  hist(census$unemploy, xlab = "Unemployment", main = "Histogram of Unemployment by County; 6 breaks", breaks = 6) hist(census$unemploy, xlab = "Unemployment", main = "Histogram of Unemployment by County; 20 breaks", breaks = 20)
  ```
  ```

Histogram breaks





Data Visualization
- Visualizing 1 Variable

Boxplots

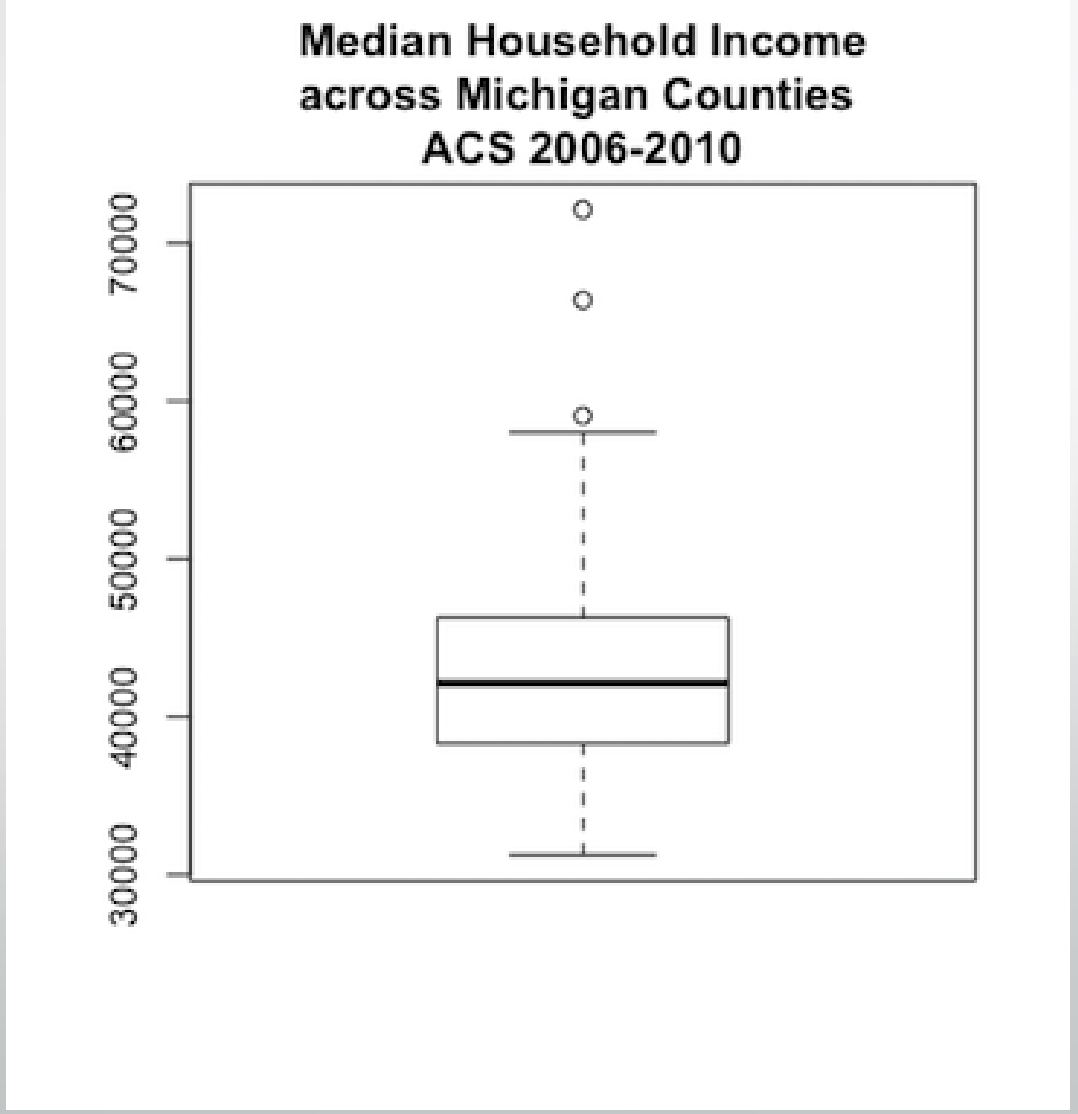
Boxplots are another useful way to show measures of distribution and can be called with the **boxplot()** function in R. Boxplots show

- the median (horizontal line in bold)
- the interquartile range (top and bottom edges of the rectangle)
- the lowest and highest values within 1.5 X
- the interquartile range (lower and upper whiskers extending from rectangle)
- and any outliers (shown as dots) in the data.

Note that a variable must be continuous to view its distribution with a boxplot.

```
boxplot(census$hh_income, main = "Median Household
Income\nacross Michigan Counties \nACS 2006-2010")
#'\n' signals a break into a new line
```

Boxplots


Median Household Income
across Michigan Counties
ACS 2006-2010
70000
60000
50000
40000
30000

Data Visualization
- Visualizing 2-3 Variables

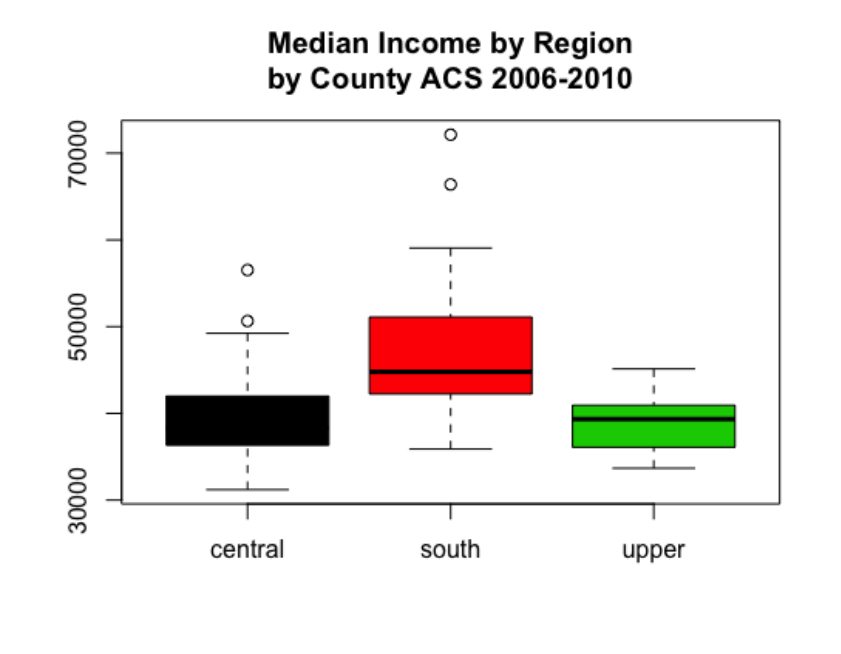
Boxplots

While we've introduced boxplots of one variable in the previous section, boxplots can also be used to show more than one variable.
As mentioned previously, to use boxplots the response variable needs to be **continuous**;
but now, when considering two variables, the predictor variable can be **discrete**.
Note the differences in syntax of your input variables in the following two examples:

Boxplots 2 variables

Example 1 Syntax: (*continuous* variable 1 ~ *discrete* variable 2)

- `boxplot(hh_income ~ region, data = census, col = palette(rainbow(3)), main = "Median Income by Region\nby County ACS 2006-2010")`
- `#this will give us household income by region`

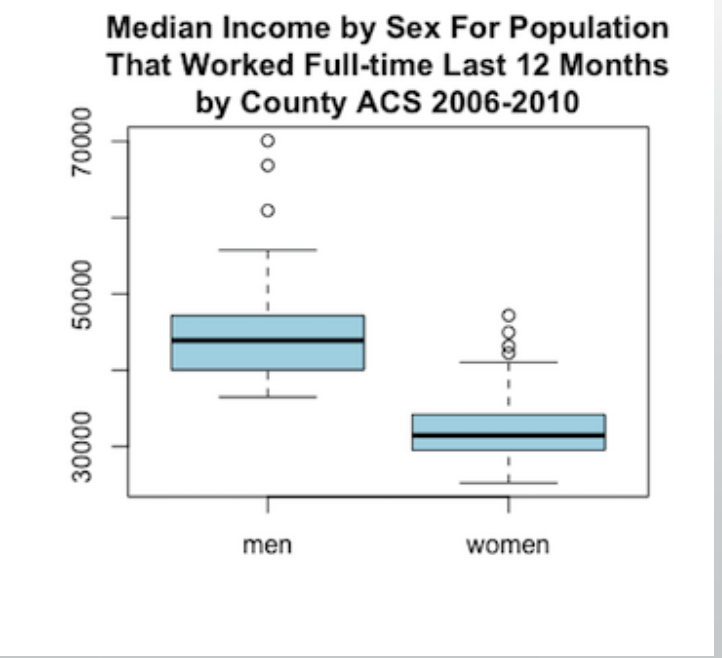


Boxplots 2 variables

Example 2 Syntax: (continuous variable 1, continuous variable 2)

```
boxplot(census[, 12], census[, 13], col =
"lightblue", names = c("men", "women"), main =
"Median Income by Sex For Population\nThat Worked
Full-time Last 12 Months\nby County ACS 2006-2010")

#selecting column 12 and 13 of the data frame 'census'
```



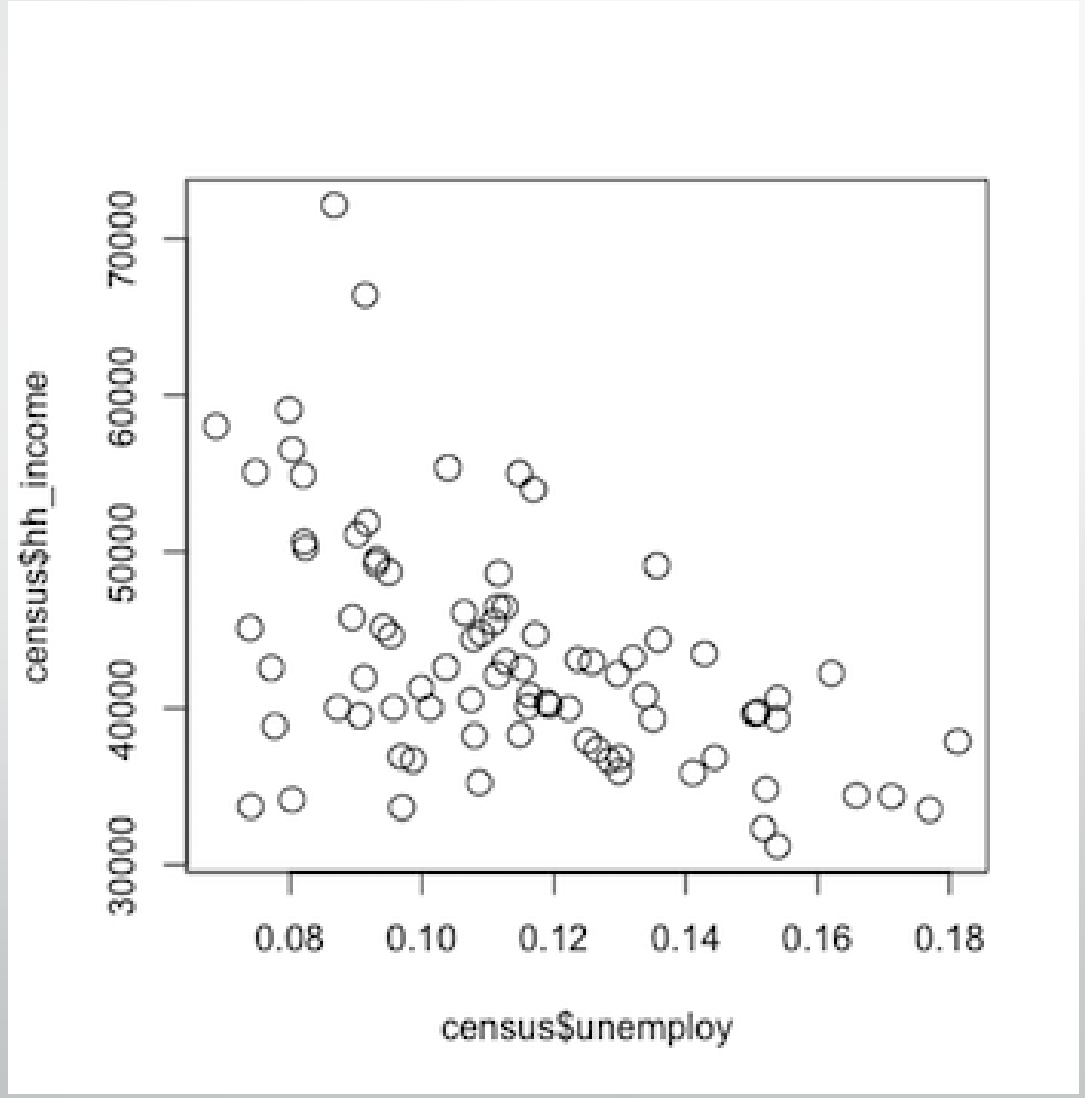
Data Visualization
- Visualizing 2-3 Variables

Scatterplots

Scatterplots are extremely common for plotting 2 **continuous** variables and can be called by using the **plot()** function. In R, enter the variable intended for the x-axis first, followed by the variable intended for the y-axis.

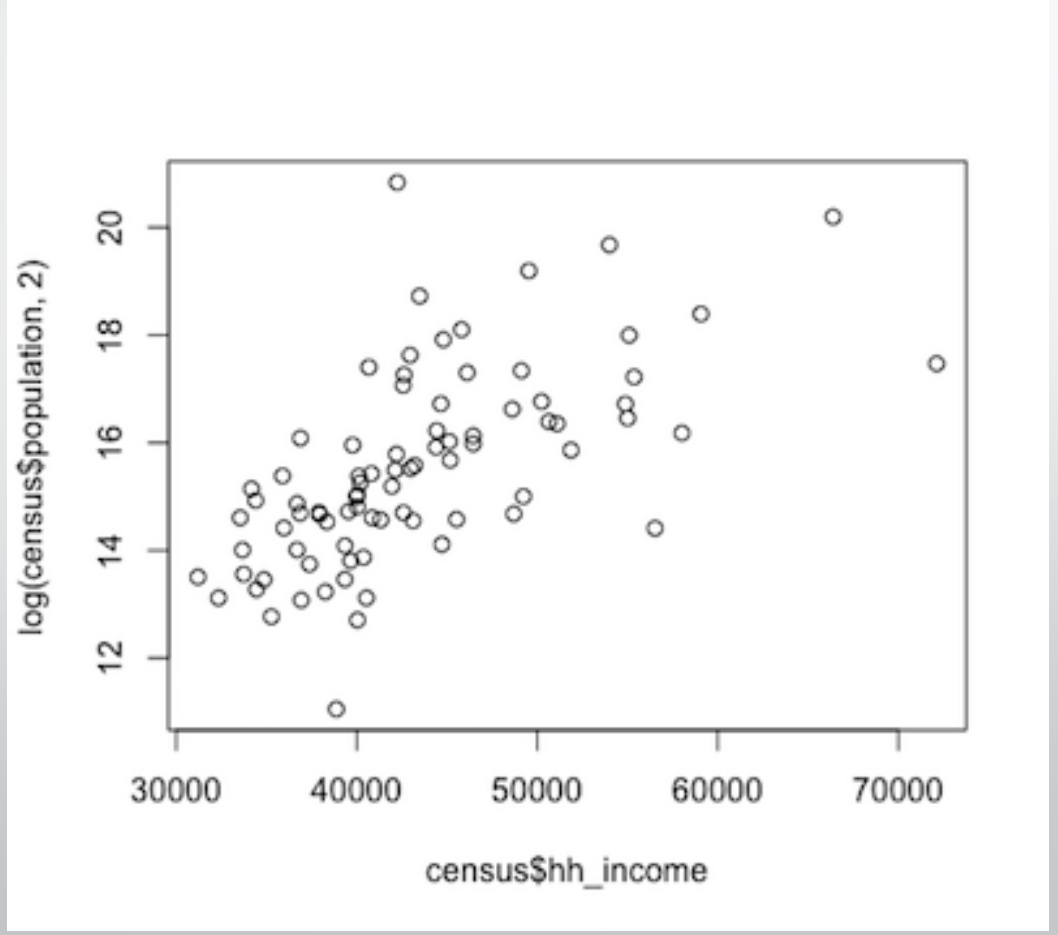
```
plot(census$unemploy, census$hh_income, cex = 1.5)
# cex controls the size of the points (1 is default)
```

Scatterplots



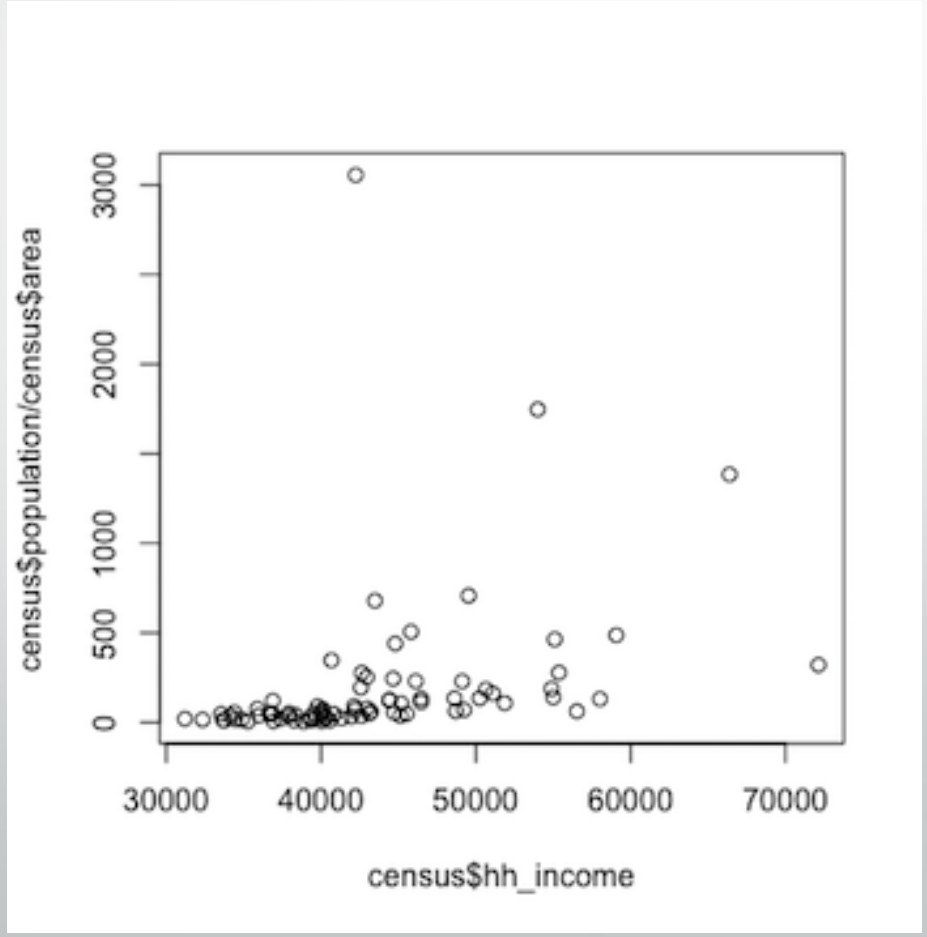
Scatterplot and Log Transformation

```
# we can plot the log of a variable by simply adding it to the plot command
plot(census$hh_income, log(census$population, 2))
```



Scatterplot and Transformation

```
# for the y axis this is the population normalized by area (population # density)
plot(census$hh_income, census$population/census$area)
```



Scatterplot - building by layers

```
plot(c(0, max(census$rent)), c(0, max(census$hh_income)), type
= "n", xlab = "Rent", ylab = "Median Household Income")
```

```
#create a blank canvas ('type' parameter) and add axis labels ('xlab' and 'ylab' parameters)
```

```
points(upper$rent, upper$hh_income, pch = 20, col = "blue")
points(central$rent, central$hh_income, pch = 20, col = "red")
points(lower$rent, lower$hh_income, pch = 20, col = "green")
legend("topleft", c("upper", "central", "lower"), pch = 20,
col = c("blue", "red", "green"), title = "Region", bty = "n")
```

```
# run '?legend' to figure out what these commands mean
```

Scatterplot - building by layers