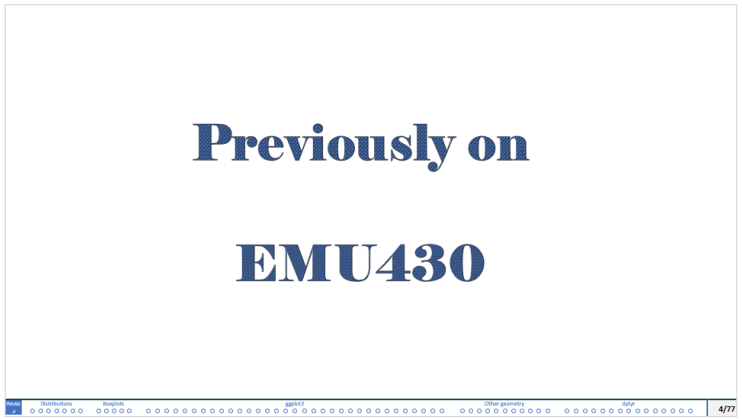
Hacettepe University
Department of Industrial Engineering
Undergraduate Program
2023-2024 Fall

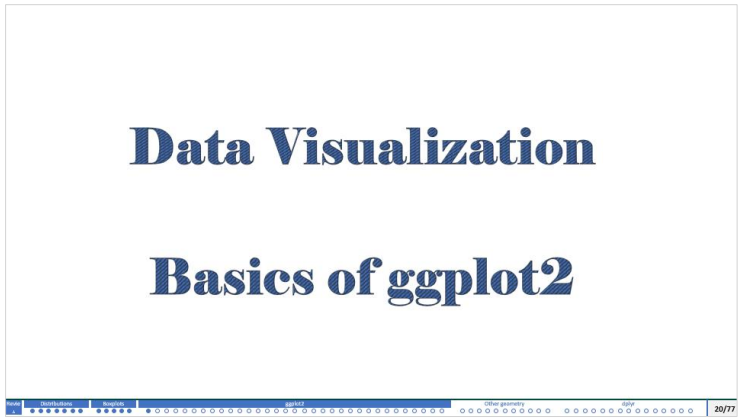
EMU 430 – Data Analytics
Week 6
November 8, 2023

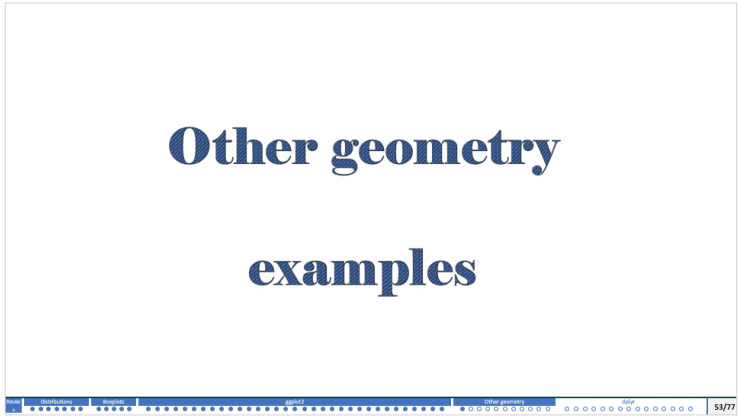
Instructor: Erdi Dasdemir

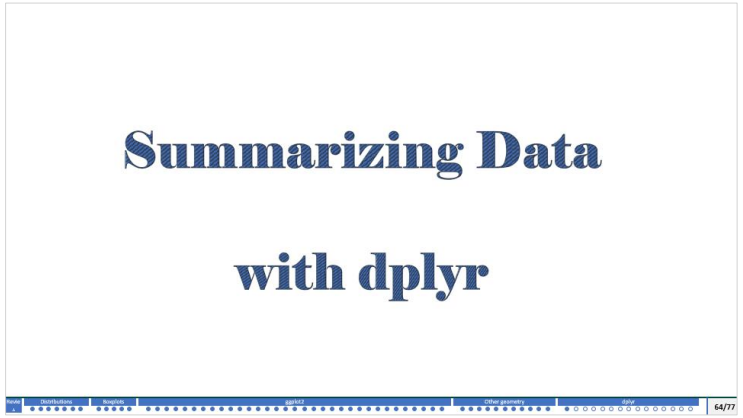
edasdemir@hacettepe.edu.tr
www.erdidasdemir.com

Outline


Previously on
EMU430


Data Visualization
Basics of ggplot2


Other geometry
examples


Summarizing Data
with dplyr

Acknowledgment

I drew inspiration primarily from Dr. Rafael Irizarry's "Introduction to Data Science" Book

and "Data Science" course by HarvardX on edX for the slides this week.

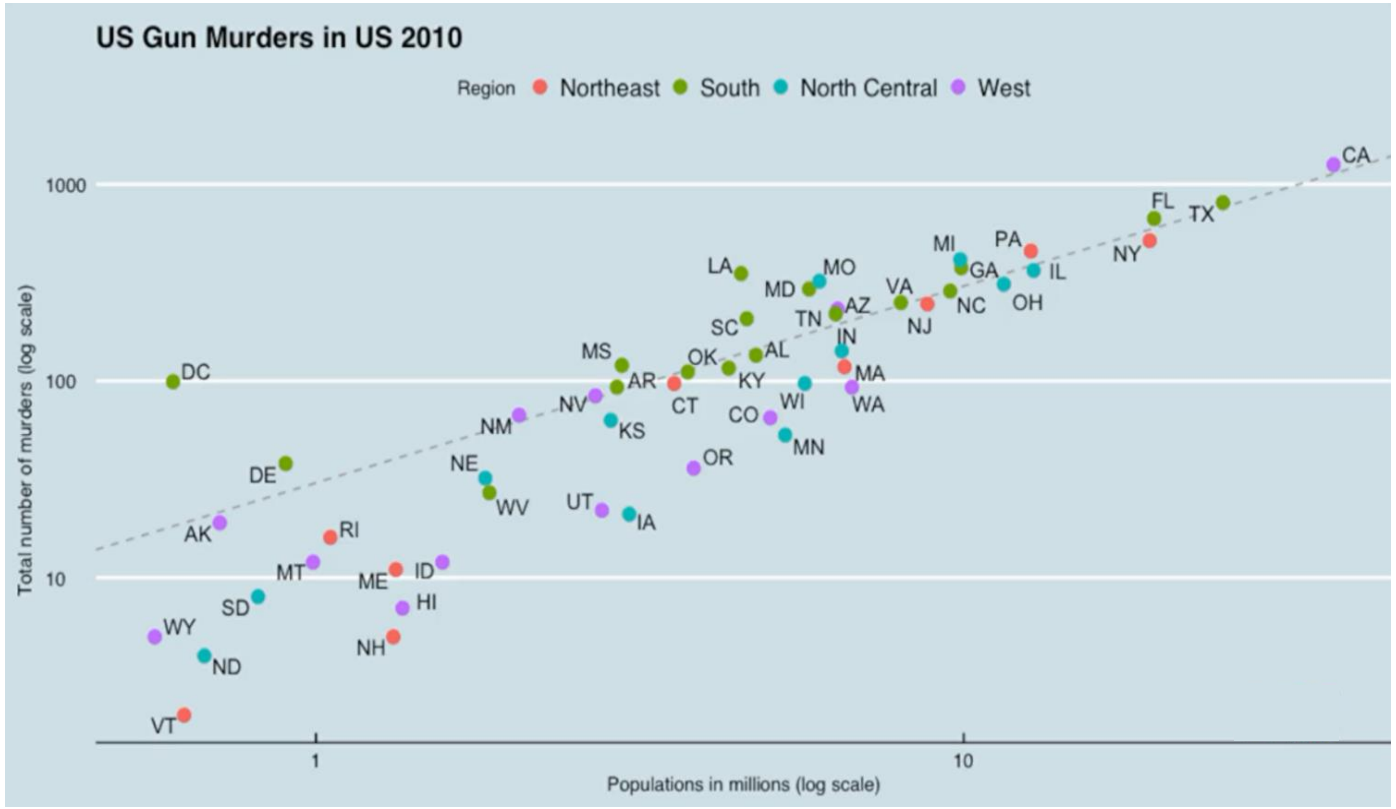
Previously on

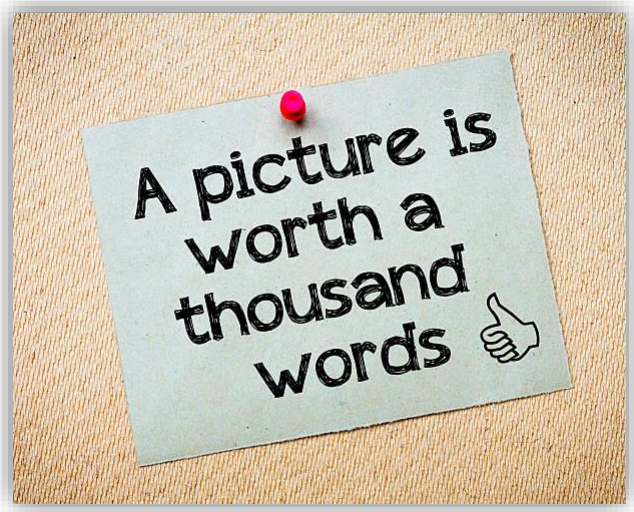
EMU430

Introduction to Data Visualization

- Numbers and character strings in a dataset is difficult to read and rarely useful.
- US murders data table.

```
> head(murders)
       state abb region population total
1    Alabama  AL  South    4779736   135
2     Alaska  AK   West     710231    19
3    Arizona  AZ   West    6392017   232
4   Arkansas  AR  South    2915918    93
5 California  CA   West   37253956  1257
6   Colorado  CO   West    5029196    65
```



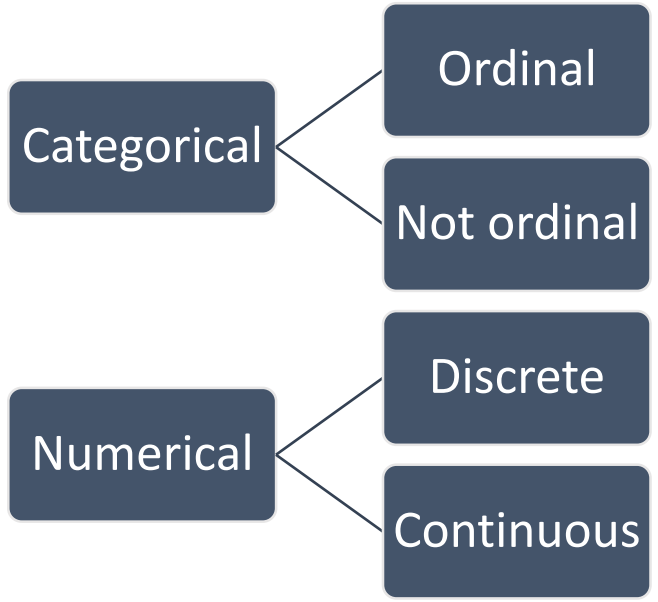


Introduction to Distributions

- Categorical: →a small number of groups
  - e.g. Regions: Northeast, South, North Central, West → **not ordinal**
  - ➢ Some categorical data can be ordered→ **ordinal data,** ex: spiciness:

    mild, medium, hot
- Numerical → Population size, murder rates, heights
  - ➢ **Continuous data:** can take any value, e.g. Heights
  - ➢ **Discrete:** population sizes (rounded numbers)

Variable types:



```
prop.table(table(heights$sex))
Female           Male
0.2266667   0.7733333
```

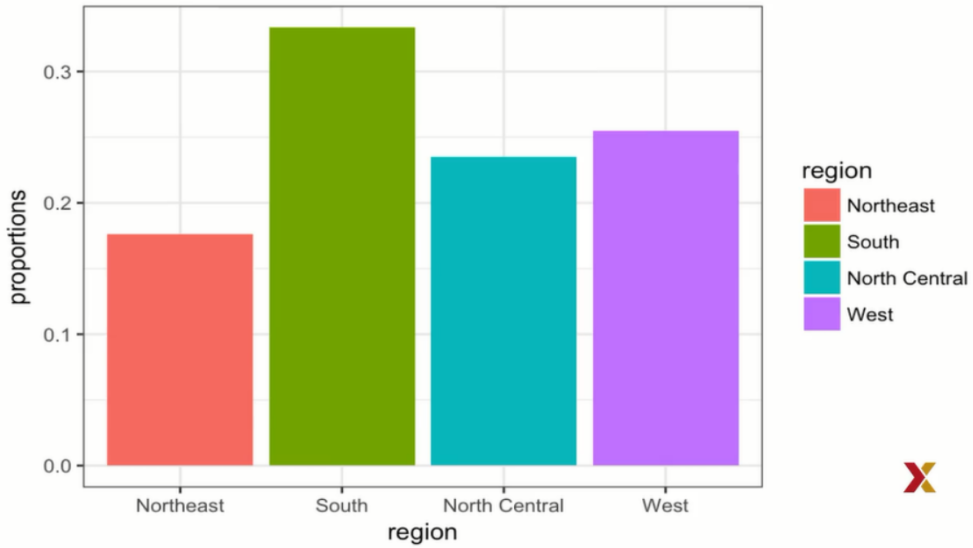
Frequency table: the simplest form of a distribution

No need to see more, a number describes all story.

Here, 23% are females and the rest are male.

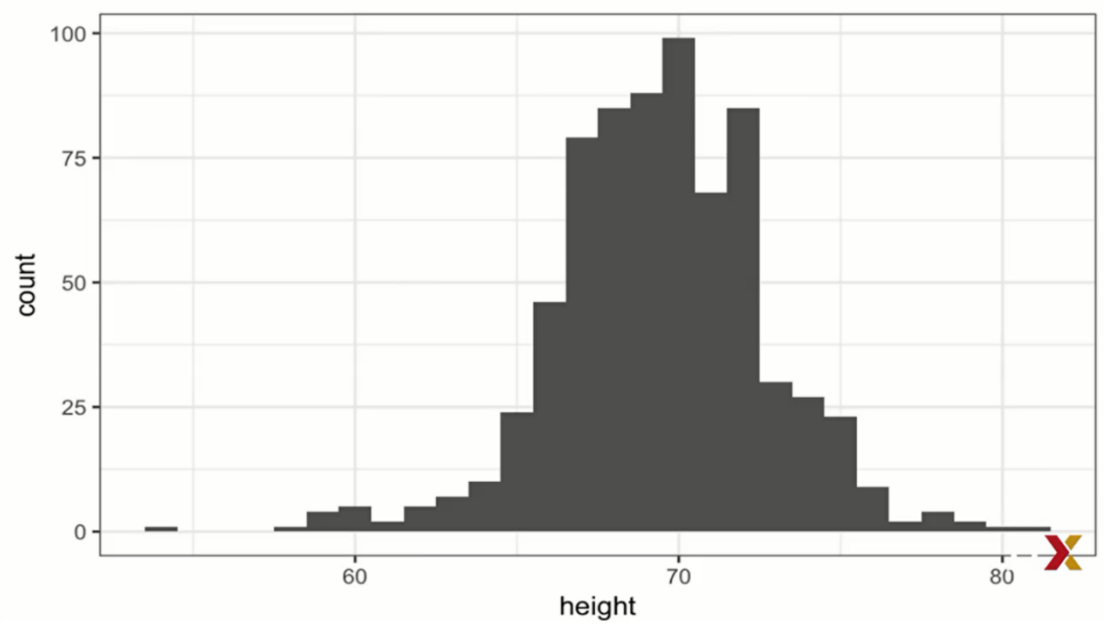
- When there are more categories, the simplest form is **barplot.**

- **Convert a vector into a visualization that summarizes all the information in the vector.**



- The simplest way to make a histogram:
  - divide a span of our data into non-overlapping bins of the same size.
  - for each bin, count the number of values that fall in that interval.
  - the histogram plots these counts as bars with the base of the bar is the interval

→ split data into 1-inch intervals

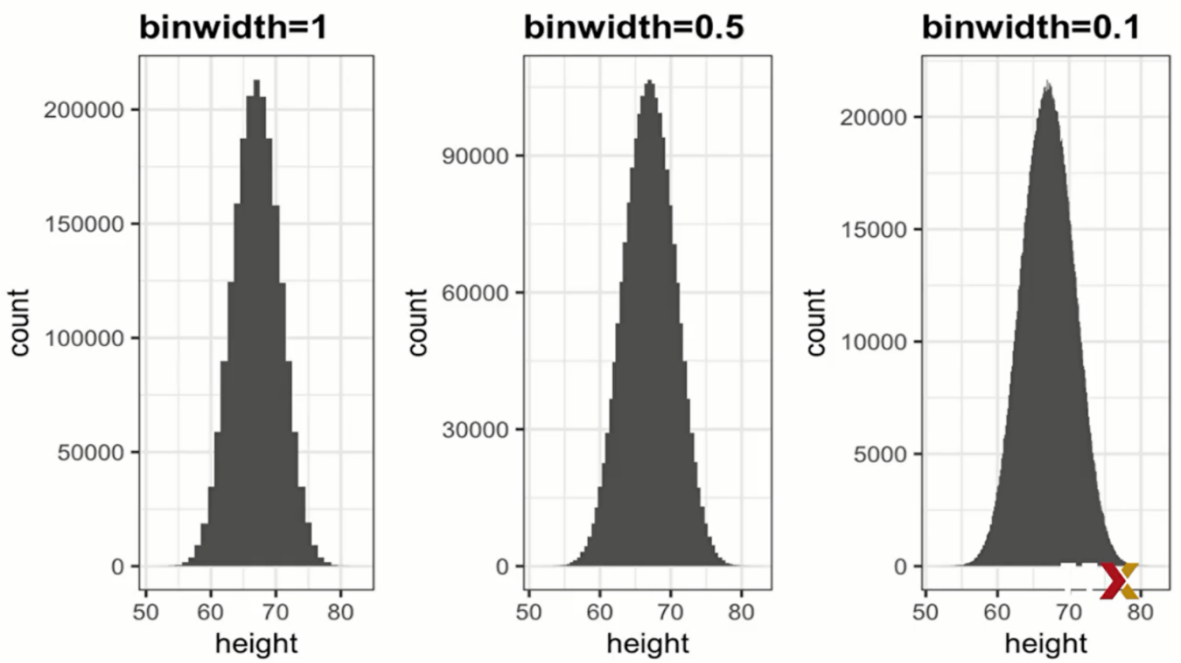


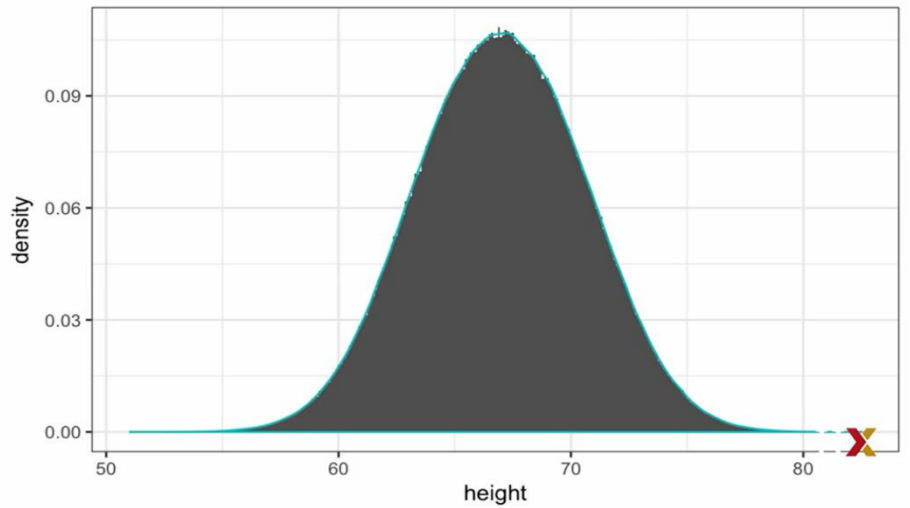
→ What we can see??

1. The range is from 55 to 81
2. More than 95% are between 63 and 75 inches.
3. Symmetric around 69 inches.
4. Proportion of data at any interval by adding up counts

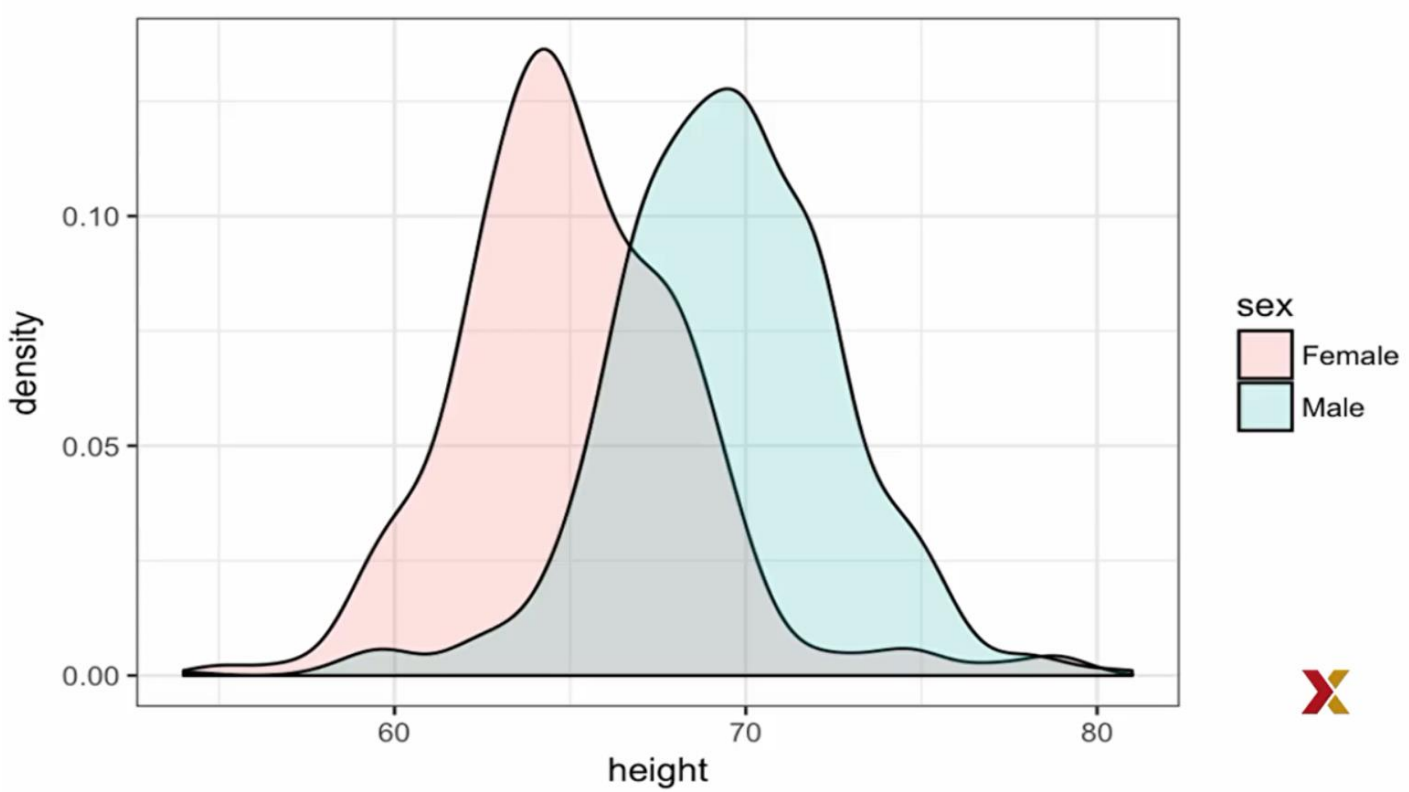
Almost all the information that we provided with CDF, and all the information in the raw data (708 heights)

- Similar to histograms but aesthetically more appealing.
- **Male heights data:**
- Assume, hypothetically, we have 1 million heights data and we can make very small bins.

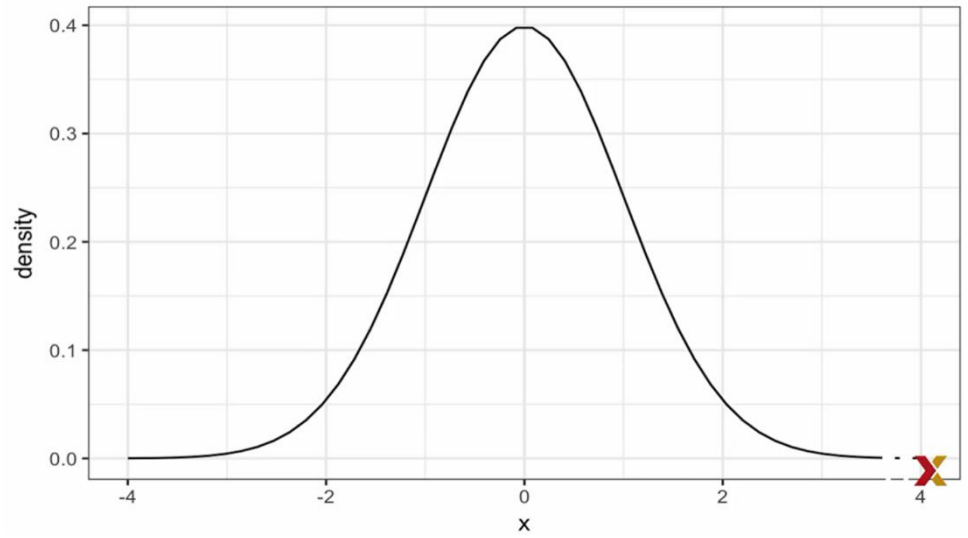




- Comparing two data sets are easier with smooth density plots compared to histograms.



- The distribution is
  - Symmetric
  - Centered at the average
  - Most values (95%) are within two standard deviations from the average



Mean → 0, standard deviation → 1

Qantiles

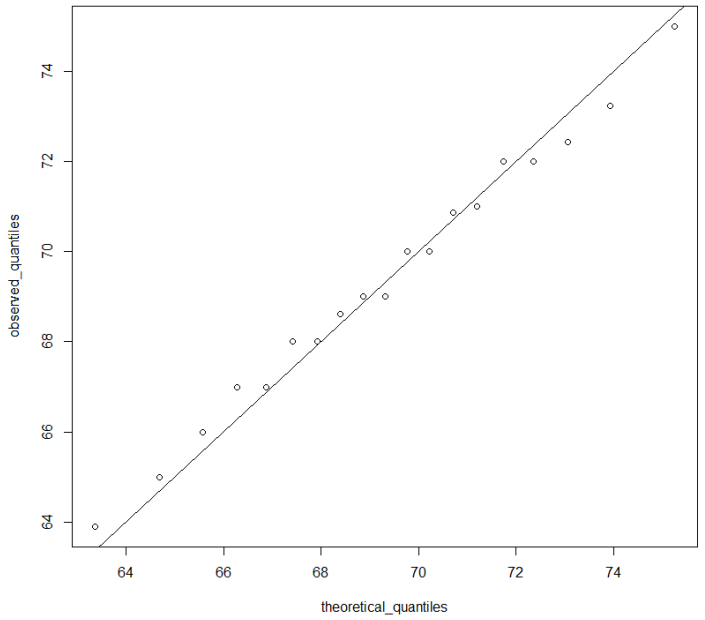
- If the quantiles for the data match the quantiles for the normal distribution → data is approximated by a normal distribution.
- Observed quantiles:

```
observed_quantiles <- quantile(x, p) # observed quantiles
```

- Theoretical normal distribution quantiles:

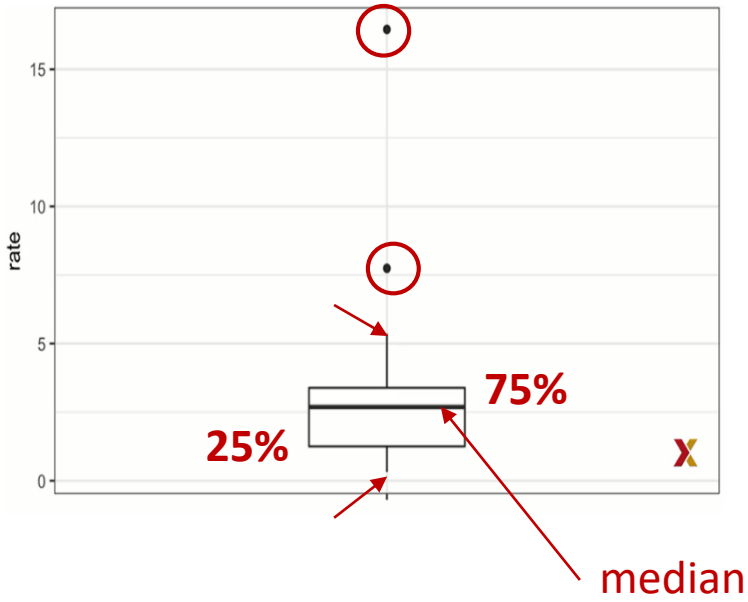
```
theoretical_quantiles <- qnorm(p, mean = mean(x), sd = sd(x)) # theoretical quantiles
```

```
p <- seq(0.05, 0.95, 0.05)
observed_quantiles <- quantile(x, p)
theoretical_quantiles <- qnorm(p, mean = mean(x), sd = sd(x))
plot(theoretical_quantiles, observed_quantiles)
abline(0, 1)
```

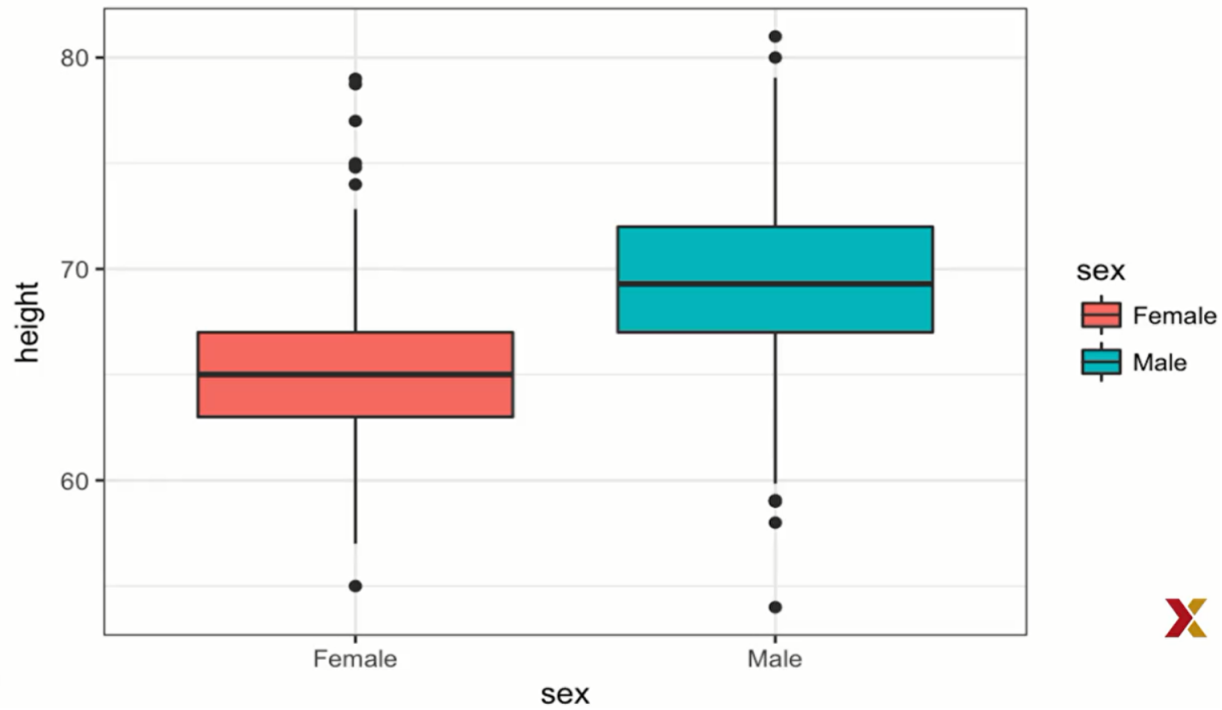


Boxplots

- John Tukey:
  - Provide a five-number summary composed of the **range** along with the quartiles (25th, 50th, 75th percentiles)
  - Ignore outliers when computing the range, plot them independently
  - Plot this as a box with whiskers



Boxplots



Exploratory Data Anaylsis Example

An Example Story

- Time to use your HI and data science skills when: “you notice something that you do not expect to see”

  - **“The greatest value of a picture is when it forces us to notice what we never expected to see.” John W. Tukey)**

Data Visualization

Basics of ggplot2

Basics of ggplot2

- R has several base functions and graphic packages to create visualizations.
- We will use ggplot2 because it breaks the graph into components.
- This permits us to create relatively complex and aesthetically pleasing plots using syntax that is intuitive and easy to remember.

Basics of ggplot2

- load the ggplot2

```
library(ggplot2)
```

- We can also load the ggplot2 package by loading `tidyverse` package.

```
library(tidyverse)
```

- `tidyverse` includes useful packages like `dplyr` in addition to `ggplot2`.

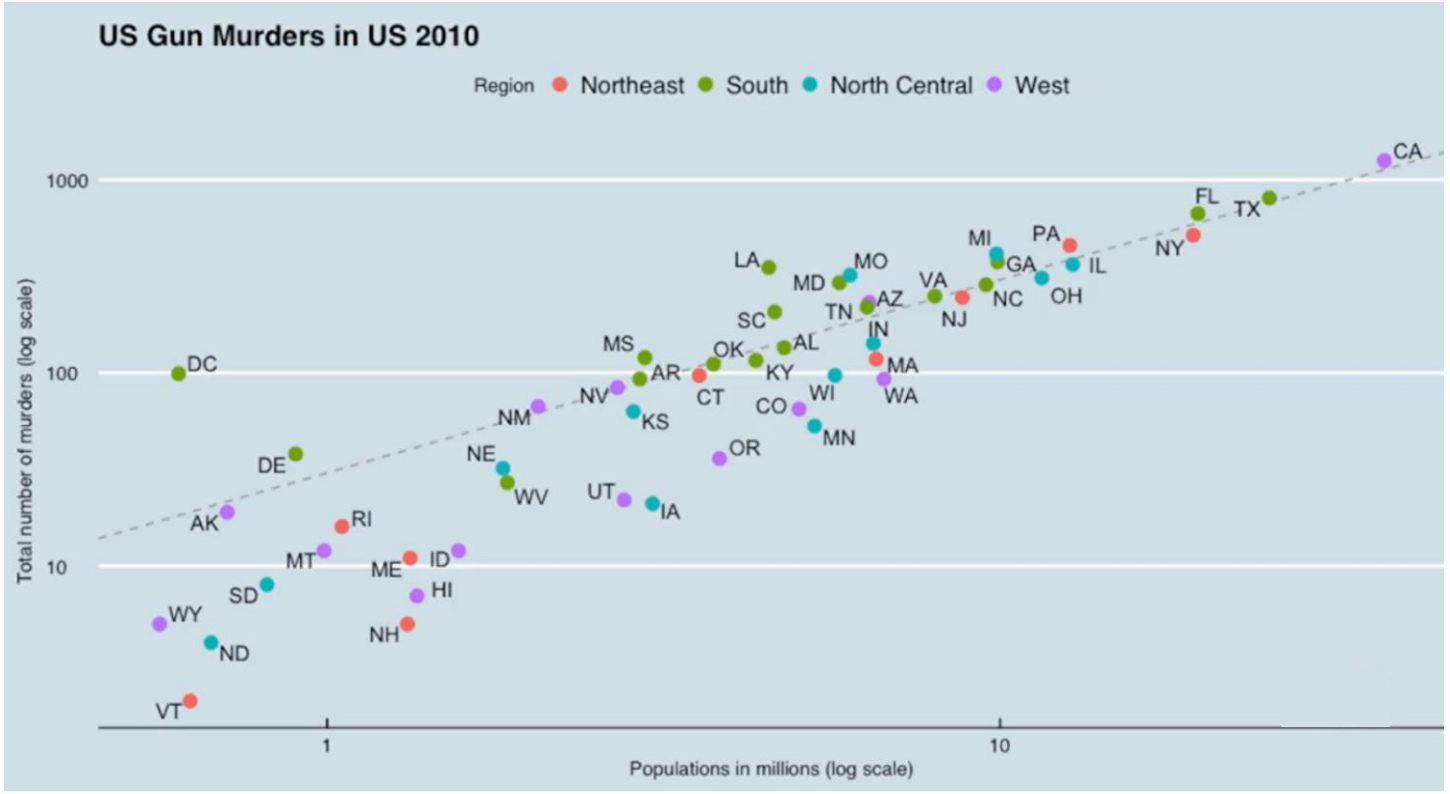
- ggplot is “grammar of graphics”.
- Learning grammar can help a beginner construct different sentences by learning just a handful of verbs, nouns, and adjectives without needing to remember all sentences.
- Similarly, we can create hundreds of plots by learning ggplot building blocks and their grammar.
- ggplot is designed to work exclusively with data tables.
  - Rows have to be observations,
  - Columns have to be variables.
  - Many data sets can be easily converted to this format.

- ggplot2 has several functions that we will use a lot.
- These are sometimes hard to remember.
- Use the cheat sheet or google whenever you need.
  - Datacamp **ggplot2 Cheat Sheet**
  - Posit (Rstudio) ggplot 2 Cheat Sheet

- We want to create a graph showing how much states vary across population size and the number of murders.

- We also want to see the relationship between murder totals and population size.

- ```
  library(tidyverse)
  library(dslabs)
  data(murders)
  ```


  ```

- **Components**

1. Data
2. Geometry components (scatter plot, bar plot, histograms, smooth densities, q-q plots, box plots)
3. Aesthetic mapping (x-axis: population size, y-axis: total number of murders, text: identify states, colors: denote four different regions
4. Scales: we generally scale ranges of the x and y-axis.
5. Labels, Title, Legend, Themes, etc. → defines style

- **Creating a new plot with Data Component**

```
library(tidyverse)
library(dslabs)
data(murders)

# first option
ggplot(data=murders)

# second option
murders %>% ggplot()
```

- **Creating a new plot with Data Component**

```
library(tidyverse)
library(dslabs)
data(murders)

# first option
ggplot(data=murders)

# second option
murders %>% ggplot()
```

Since we did not assign it to a variable, the plot is shown automatically. We can also assign it to a variable:

```
p <- murders %>% ggplot()
class(p)
[1] "gg"     "ggplot"
print(p)
p
```

Layers

- In ggplot, we create graphs by adding layers.
- We add them component by component.
- Layers can define geometries, compute summary statistics, define scales, change styles

- To add layers we use +
- In general, a line of code in ggplot will look like this:

```
data %>% ggplot() + layer 1 + layer 2 + ... + layer n
```

Layers

- In general, a line of code in ggplot will look like this:

```
data %>% ggplot() + layer 1 + layer 2 + ... + layer n
```

- Usually the first added layer defines the geometry.
- We want to use scatter plot. What geometry do we use?
- We can go to help or ask google. The answer is `geom_point`.
- This is a general form. We create geometry with `geom_xxxx`

Layers: Geometry and Aesthetic Mapping

- In general, a line of code in ggplot will look like this:

```
data %>% ggplot() + layer 1 + layer 2 + ... + layer n
```

- For `geom`, we need to provide **data** and **mapping**.

```
?geom_point()
```

data

```
p <- murders %>% ggplot()
```

mapping

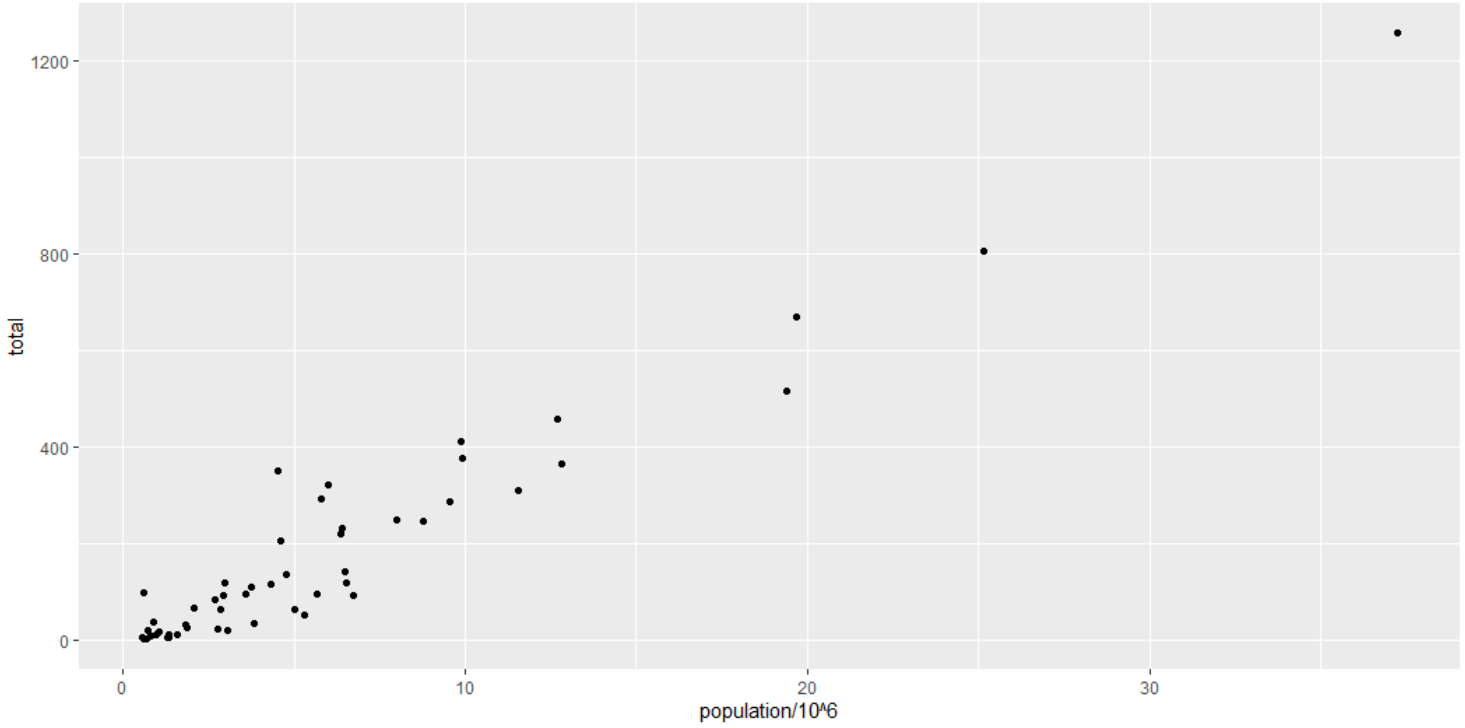
aes: this function connects data with what we see on the graph. we will use this frequently.

aesthetic mapping:

```
murders %>% ggplot() + geom_point(aes(x = population/10^6, y = total))
```

Layers

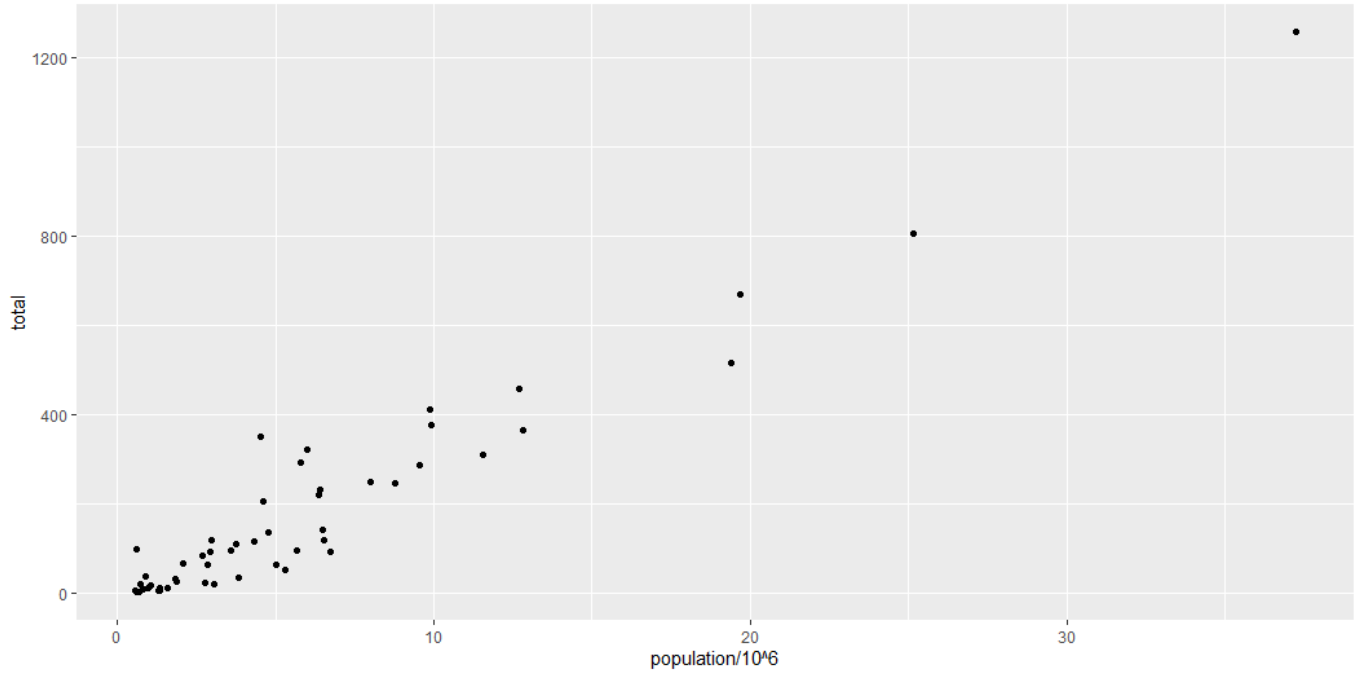
```
murders %>% ggplot() + geom_point(aes(x = population/10^6, y = total))
```



Layers

- We can add layers to previously defined objects.

```
p <- ggplot(data = murders)
p <- murders %>% ggplot()
p + geom_point(aes(x = population/10^6, y = total))
```



Layers: Scales and Labels

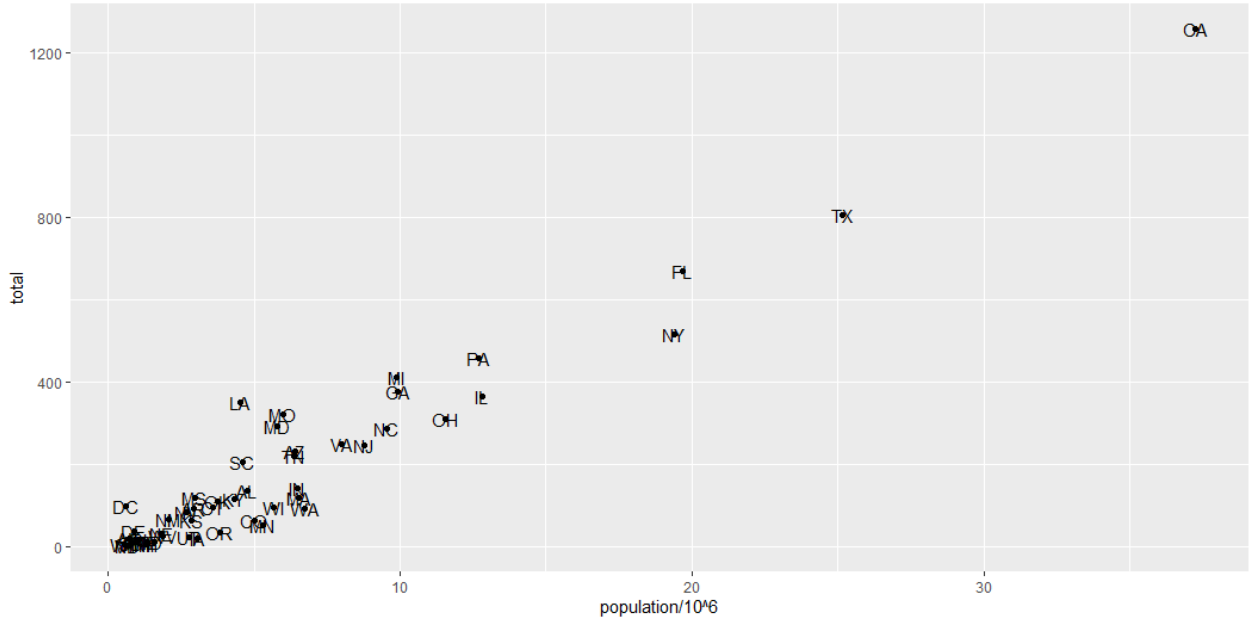
```
p + geom_point(aes(x = population/10^6, y = total))
```

- Scales and labels are defined by default when adding above the first layer.
- The second layer in the plot we want to add involves adding a label to each point.
  - `geom_label` and `geom_text` functions permit us to add text to the plot.
  - Because each state, each point, has a label, we need an aesthetic mapping to make this connection.

```
p + geom_point(aes(x = population/10^6, y = total)) +
  geom_text(aes(population/10^6, total, label = abb))
```

Layers: Scales and Labels

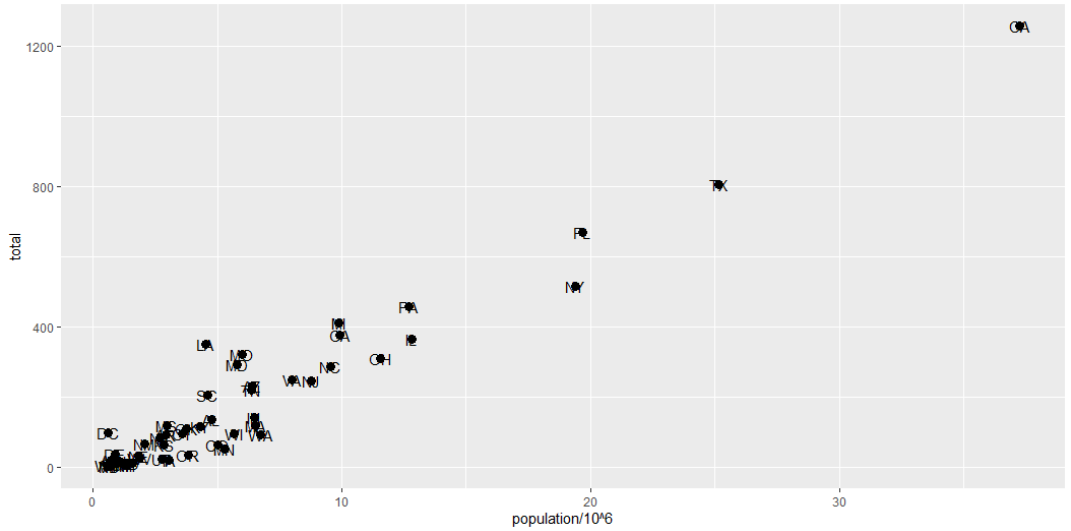
```
p + geom_point(aes(x = population/10^6, y = total)) +
  geom_text(aes(population/10^6, total, label = abb))
```



Layers: Tinkering

- Each geometry function has many arguments other than data and aes.
- if you to help of `geom_point`, for example, there is `size`

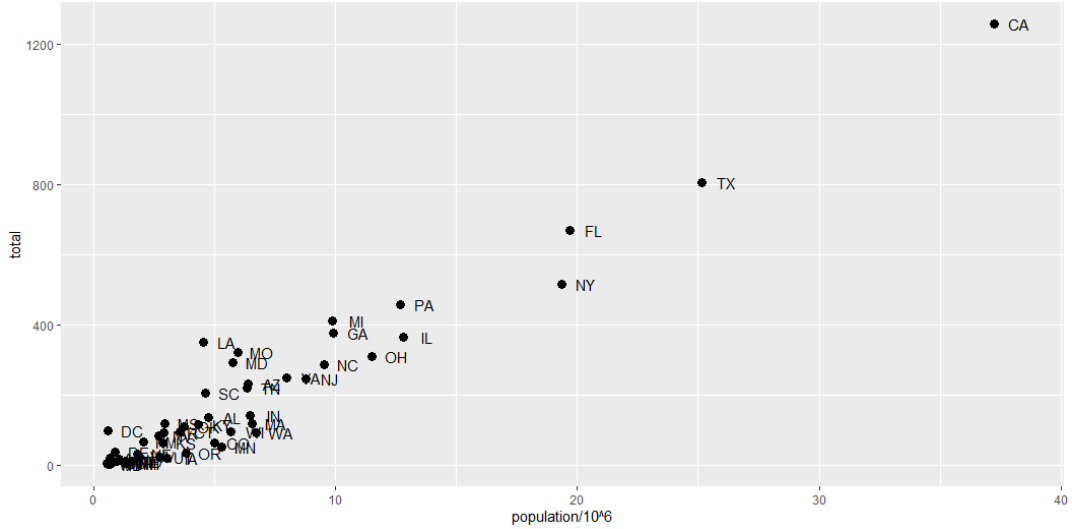
```
p + geom_point(aes(x = population/10^6, y = total), size = 3) +
geom_text(aes(population/10^6, total, label = abb))
```



Layers: Tinkering

- Now the points are larger, we cannot read labels.
- if we go to help of `geom_text`, there is an argument called `nudge_x`

```
p + geom_point(aes(x = population/10^6, y = total), size = 3) +
geom_text(aes(population/10^6, total, label = abb), nudge_x = 1)
```



- We can make the code more efficient.
- We have been mapping the population and total to the points twice.
- We can avoid this by adding what is called a `global aesthetic` mapping.

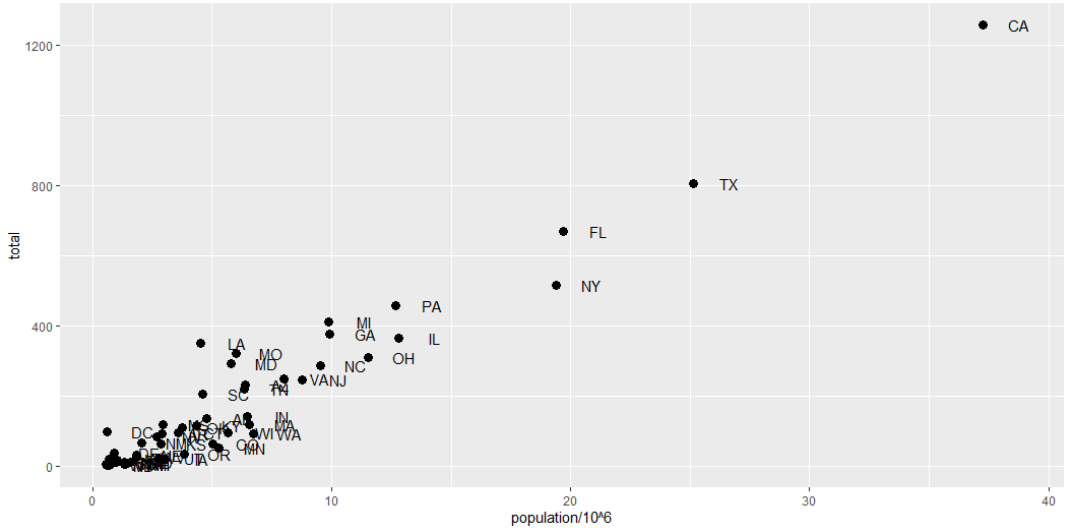
```
args(ggplot)
```

```
function (data = NULL, mapping = aes(), ..., environment = parent.frame())
```

```
NULL
```

- If we define a mapping in ggplot, then all the geometries that are added as layers will default to this mapping.

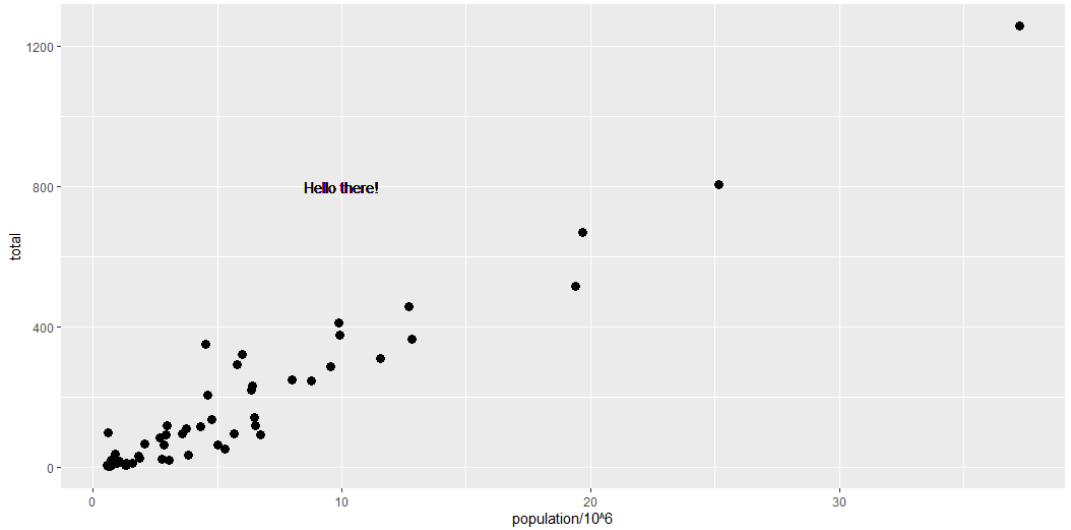
```
p <- murders %>% ggplot(aes(population/10^6, total, label = abb))
p + geom_point(size = 3) + geom_text(nudge_x = 1.5)
```



- We can override global mappings by local mappings.
- The flexibility of being able to redefine mappings in each layer is very useful.

```
p <- murders %>% ggplot(aes(population/10^6, total, label = abb))
```

```
p + geom_point(size = 3) + geom_text(aes(x = 10, y = 800, label = "Hello there!"))
```



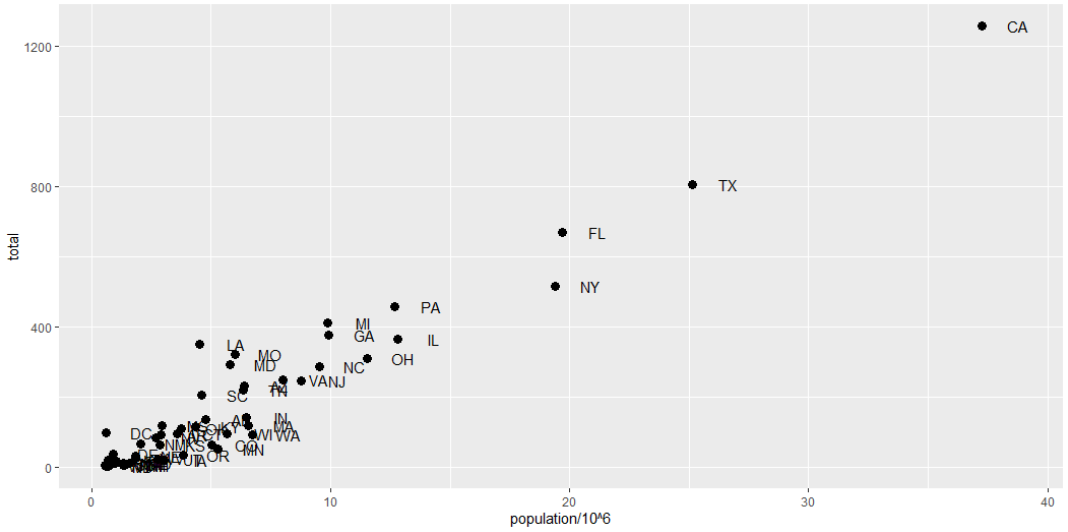
Layers: Scales, labels and colors

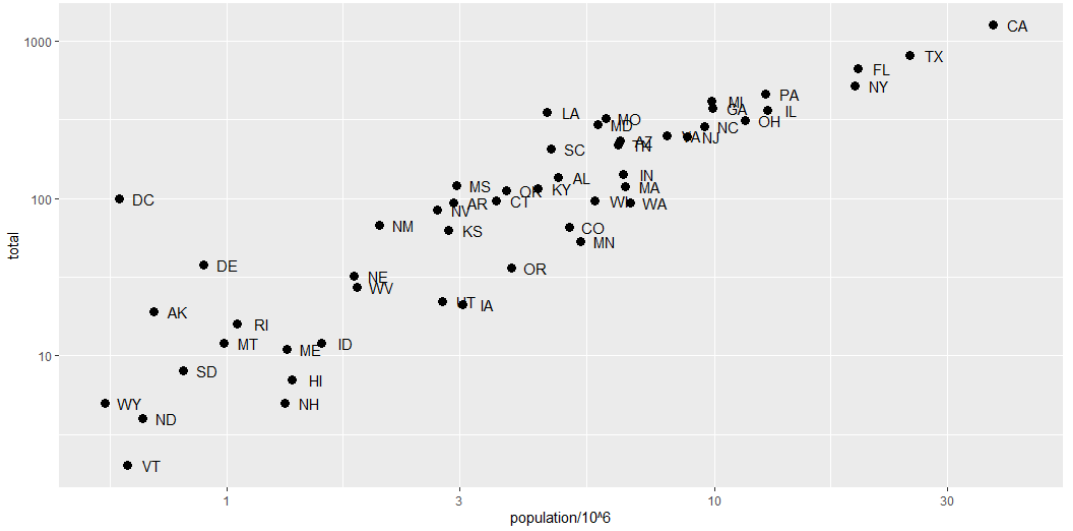
- adjust scales of axes
- change labels of axes
- add colors
- add a line

First, our desired scales are in the log scale.

```
p <- murders %>%
ggplot(aes(population/10^6, total, label
= abb))

p + geom_point(size = 3) +
  geom_text(nudge_x = 0.05) +
  scale_x_continuous(trans = "log10") +
  scale_y_continuous(trans = "log10")
```

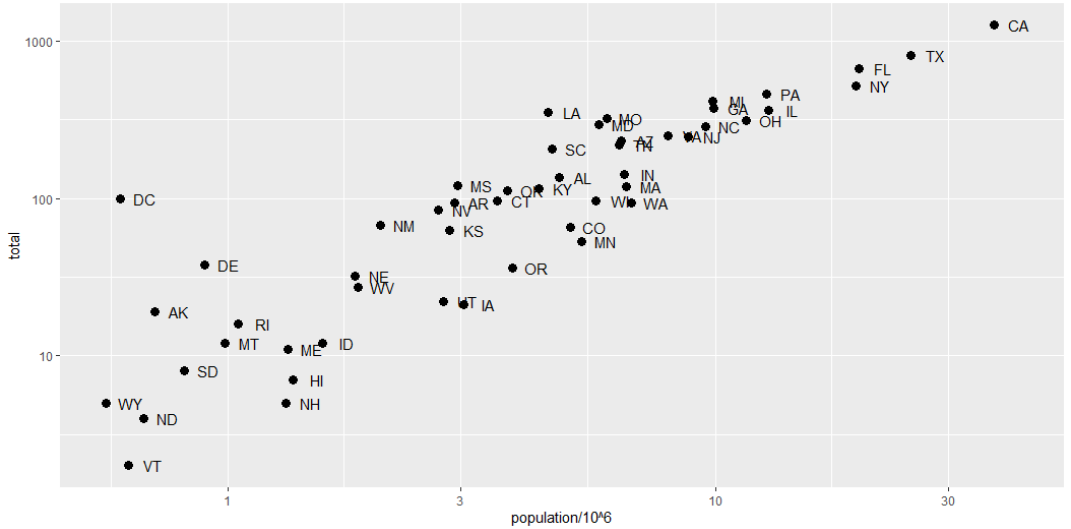




Layers: Scales, labels and colors

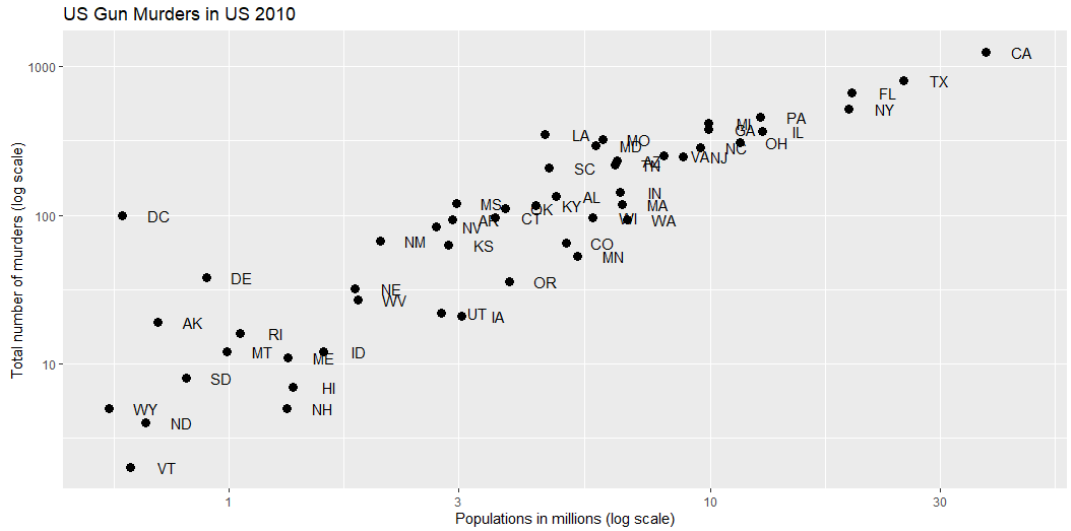
- Log transformation is so common, that ggplot provides specialized functions.

```
p + geom_point(size = 3) +
  geom_text(nudge_x = 0.075) +
  scale_x_log10() +
  scale_y_log10()
```



Add labels to x and y axis

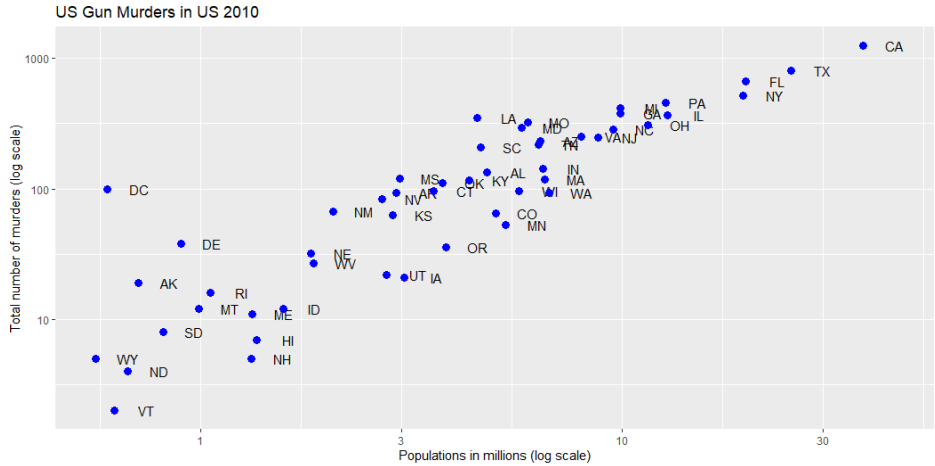
```
p + geom_point(size = 3) +
  geom_text(nudge_x = 0.075) +
  scale_x_log10() +
  scale_y_log10() +
  xlab("Populations in millions (log scale)") +
  ylab("Total number of murders (log scale)") +
  ggtitle("US Gun Murders in US 2010")
```



Change colors

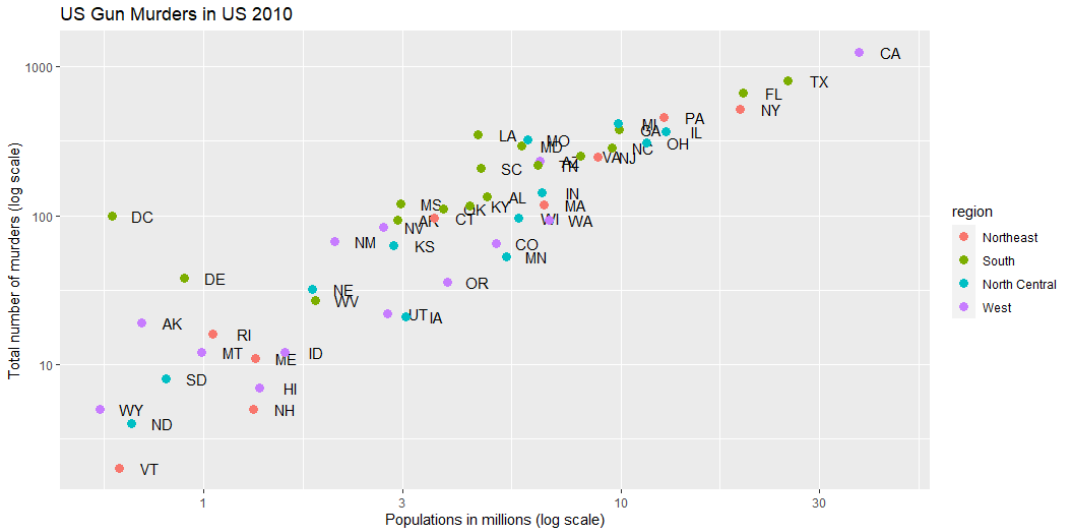
```
p <- murders %>% ggplot(aes(population/10^6,
total, label = abb)) +
  geom_text(nudge_x = 0.075) +
  scale_x_log10() +
  scale_y_log10() +
  xlab("Populations in millions (log scale)") +
  ylab("Total number of murders (log scale)") +
  ggtitle("US Gun Murders in US 2010")


# make colors all blue
p + geom_point(size = 3, color = "blue")
```



Change colors

- We want the colors to be associated with their geographical region.
- If we assign a categorical variable to the color argument, it automatically assigns a different color to each category.
- As the color of each point will depend on the category and the region from which each state is, we have to use mapping.
- To map each point to a color, we need to use `aes` since this is a mapping.




```
# make colors all blue
p + geom_point(aes(col = region), size = 3)
```

Legends

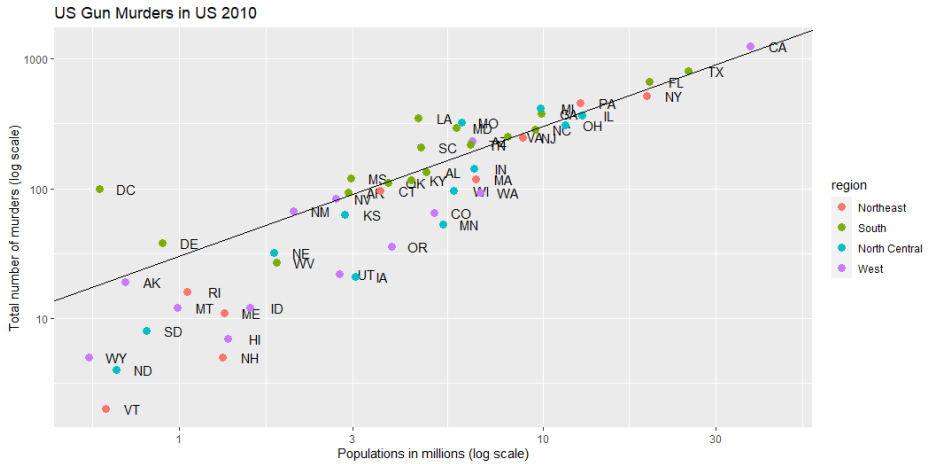
- Default behavior of ggplot automatically adds a legend that maps colors to region.

Line

- Finally, we want to add a line that represent the average murder rate for the entire country.

```
r <- murders %>% summarize(rate = sum(total) / sum(population)*10^6) %>% .$rate
# to add a line we use geom_abline
# intercept a and slope b
# default has slope 1, intercept 0

p + geom_point(aes(col = region), size = 3) + geom_abline(intercept = log10(r))
```



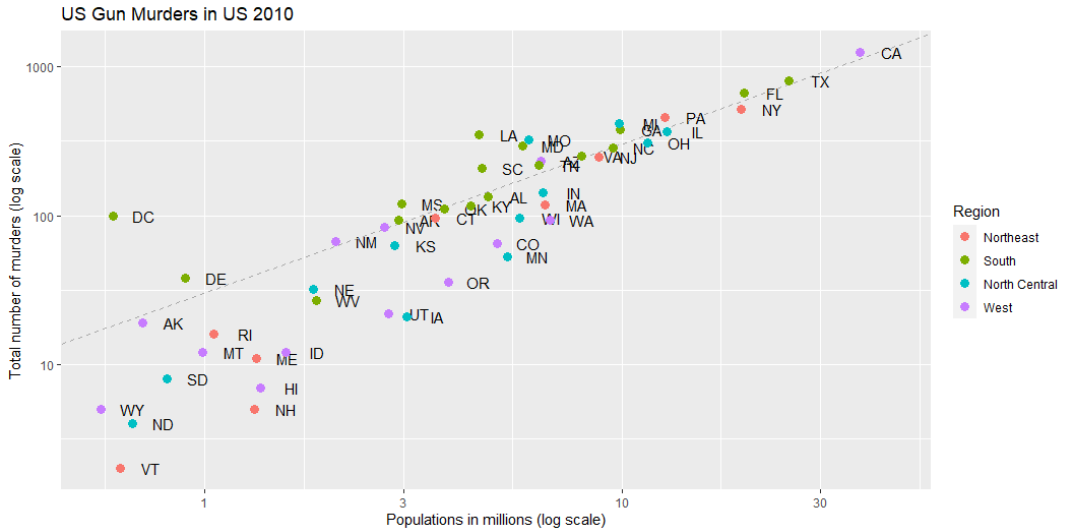
Line

- We can change line style

```
p + geom_abline(intercept = log10(r), lty = 2, color = "darkgrey") +
geom_point(aes(col=region), size = 3)
```

- keep in mind! ggplot is very flexible! there is almost always a way to achieve what you want.
- For example, you want to capitalize the legend title 'region', add a layer for this, `scale_color_discrete,`

```
p <- p + scale_color_discrete(name = "Region")
```



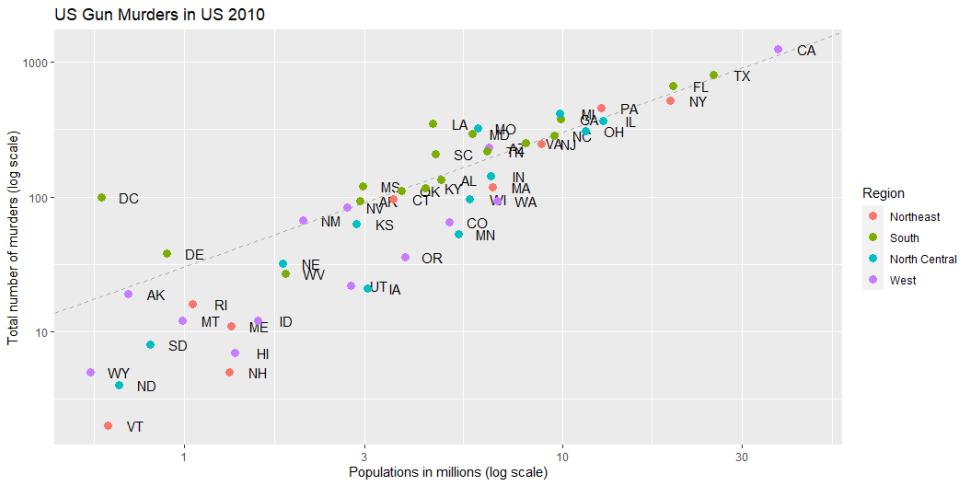
- The power of ggplot2 is further augmented thanks to the availability of add on packages.
- Finishing touches on our graph requires `ggthemes` and `ggrepel`.
- Style of a ggplot graph can be changed using the `theme` function.
- Several themes are included as part of ggplot2 package.
- There is `ggthemes` packages where many other themes are available. For example, `theme_economist`
- We can change the style by adding a layer.

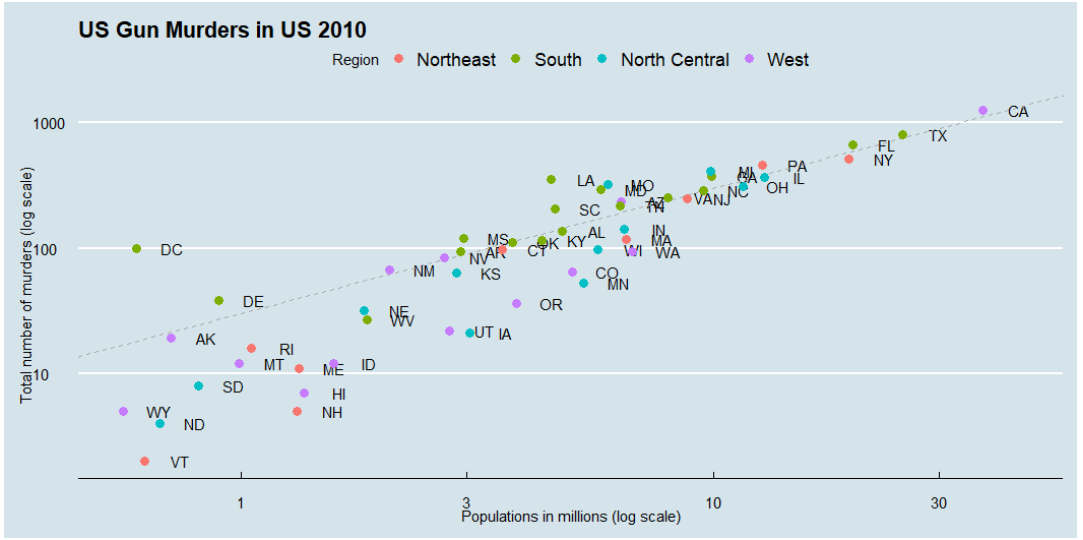
- We can change the theme style by adding a layer.

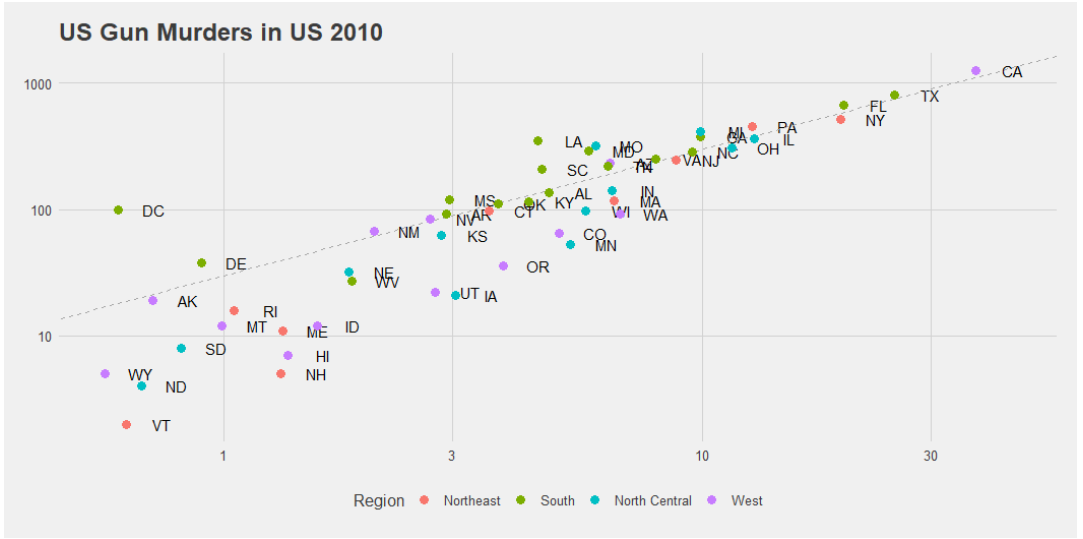
```
library(ggthemes)

p + theme_economist()

p + theme_fivethirtyeight()
```







- Change the positions of labels so that they do not fall on top of each other.
- We will use `ggrepel`.
- `ggrepel` includes a geometry that ensures they do not fall on top of each other.
- So all we need to do is change the `geom_text` layer with a `geom_text_repel` layer.

- **Start from scratch**

```
library(ggthemes)
library(ggrepel)

### first define the slope of the line
r <- murders %>% summarize(rate = sum(total) / sum(population) * 10^6) %>% .$rate

## now make the plot.
murders %>% ggplot(aes(population/10^6, total, label = abb)) +
    geom_abline(intercept = log10(r), lty = 2, color = "darkgrey") +
    geom_point(aes(col = region), size = 3) +
    geom_text_repel() +
    scale_x_log10() +
    scale_y_log10() +
    xlab("Populations in millions (log scale)") +
    ylab("Total number of murders (log scale)") +
    ggtitle("US Gun Murders in US 2010") +
    scale_color_discrete(name = "Region") +
    theme_economist()
```

Other geometry examples

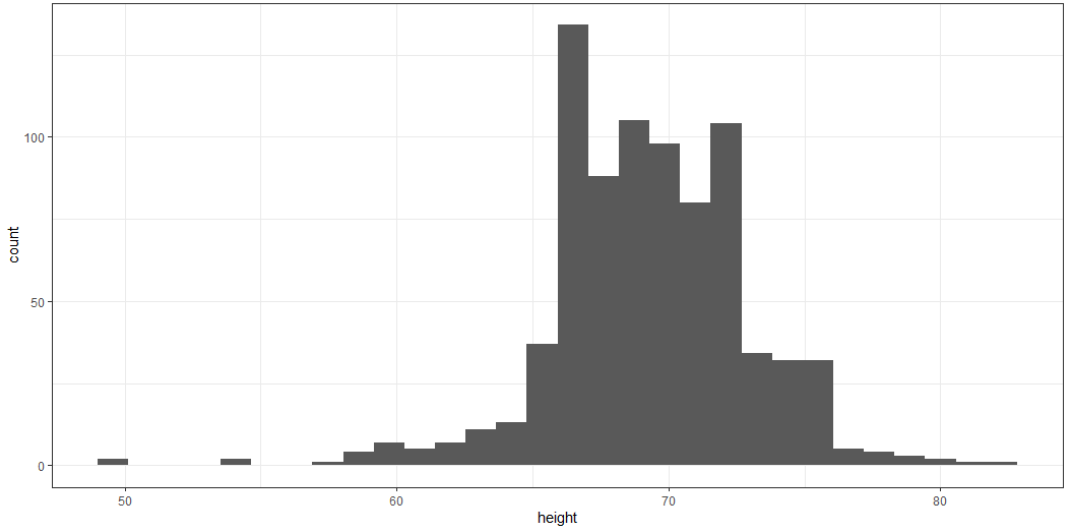
historgram

Let's make the histogram for the male heights.

```
p <- heights %>% filter(sex == "Male")
```

```
p <- p %>% ggplot(aes(x = height))
```

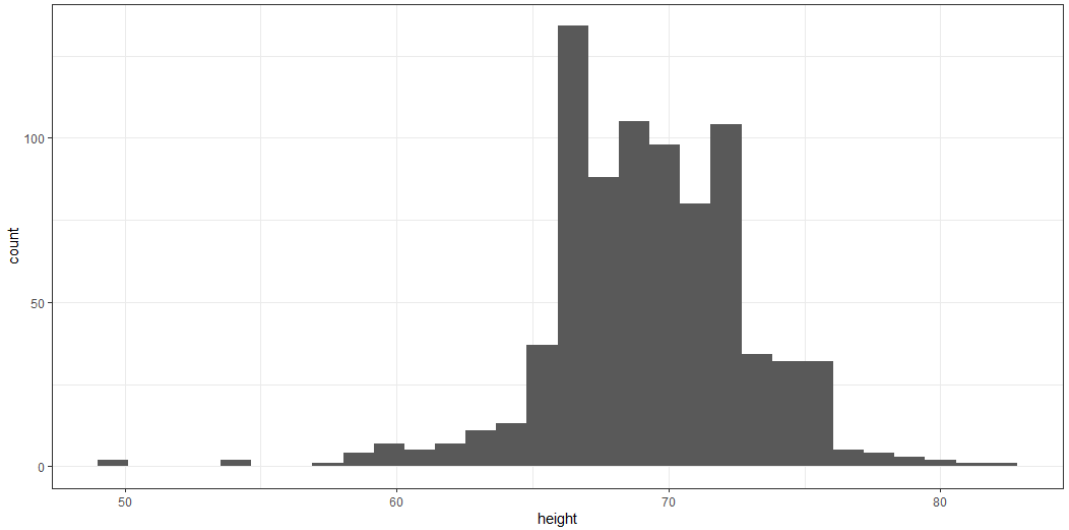
```
p + geom_histogram()
```

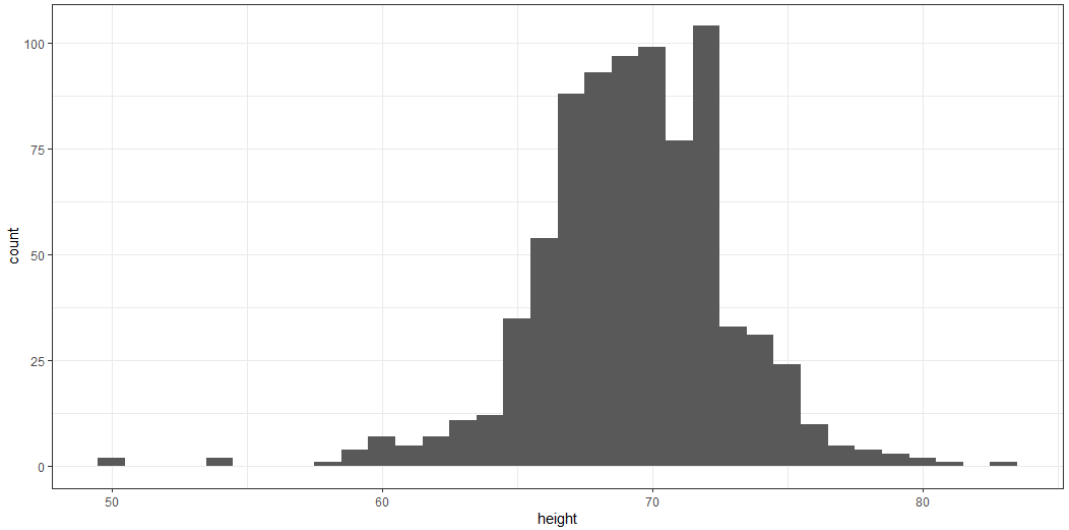


historgram

- We can define our own bin width.
- will set bin width to 1.

```
p  + geom_histogram(binwidth = 1)
```

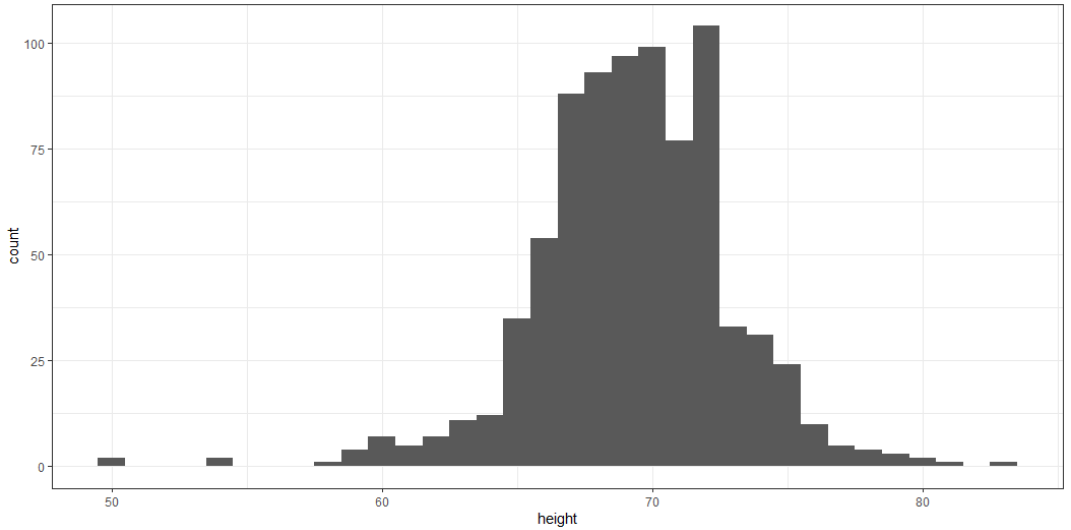


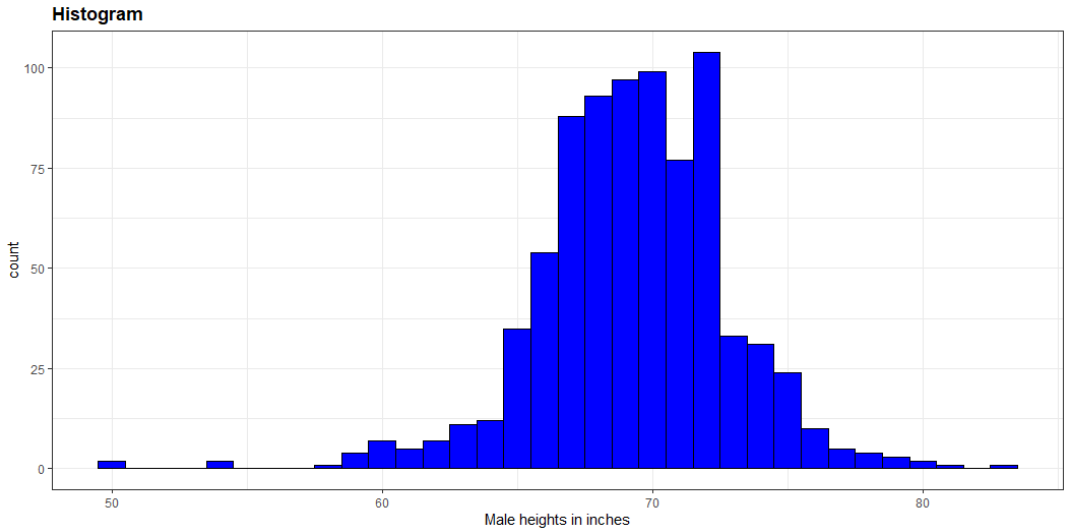


historgram

We can modify color and customize the plot more.

```
p + geom_histogram(binwidth = 1, fill =
"blue", col = "black") + xlab("Male heights
in inches") + ggtitle("Histogram")
```





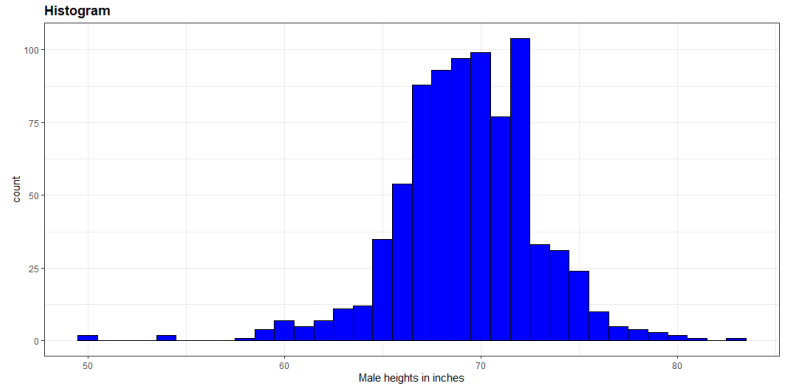
Smooth density

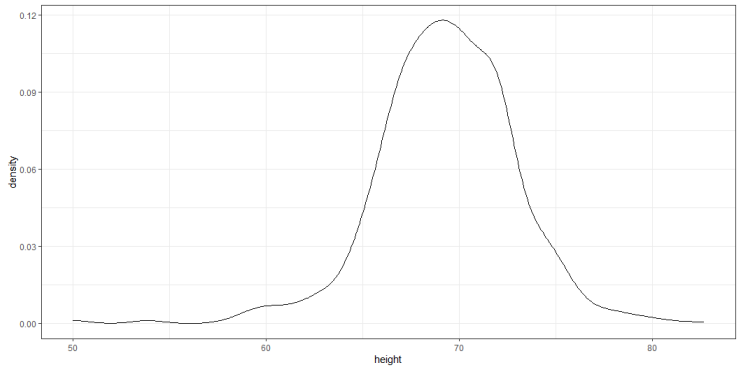
We can use geom_density() geometry to create smooth densities.

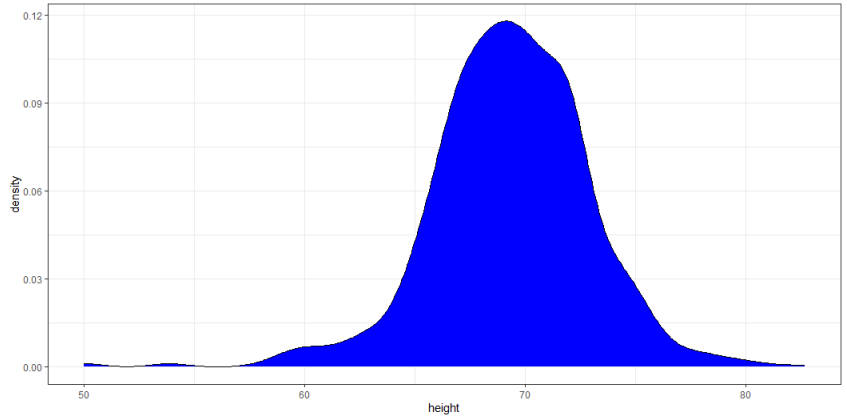
```
p + geom_density()
```

We can add color by using the fill argument.

```
p + geom_density(fill="blue")
```







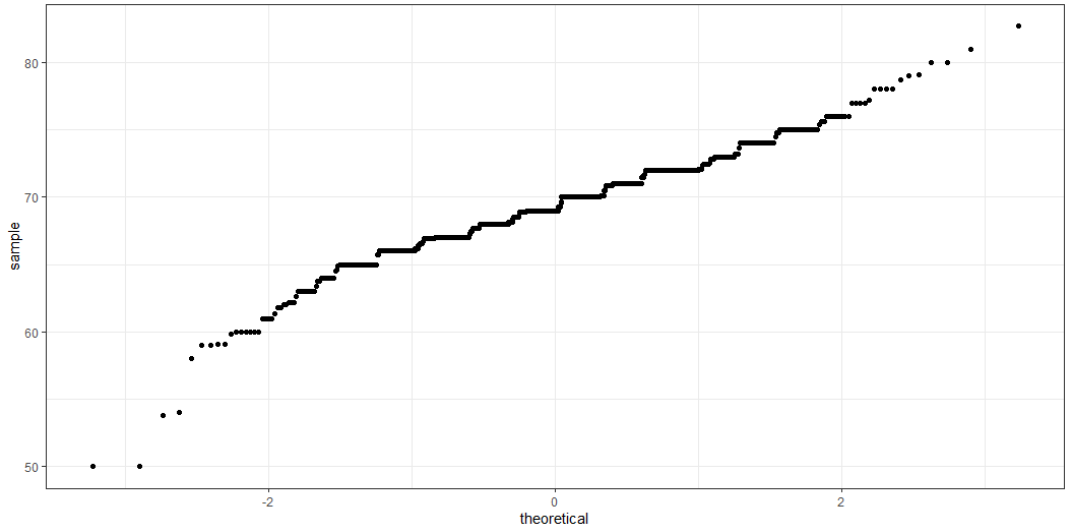
- **Q-Q plot**we can use `geom_qq()` geometry.
- Help file says that we need to specify the sample argument. Sample corresponds to our data.

```
p <- heights %>% filter(sex == "Male") %>%

ggplot(aes(sample = height))



p + geom_qq()
```

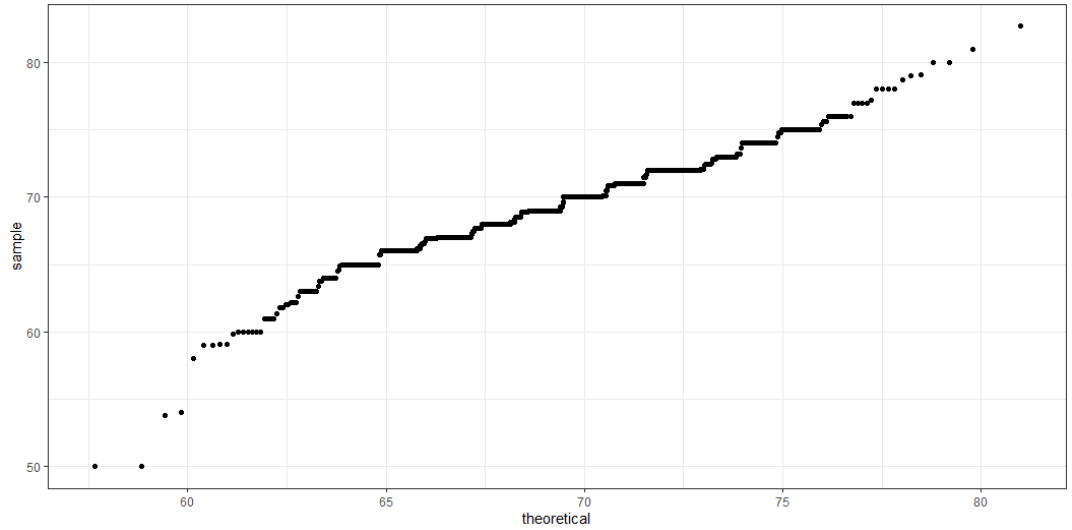


- By default, the Q-Q plot is compared to the normal distribution with average zero and standard deviation 1.
- We need to use `dparams` argument to change this.

```
params <- heights %>% filter(sex == "Male") %>%
summarize(mean = mean(height), sd = sd(height))
params
```

```
      mean       sd
1 69.31475 3.611024
```

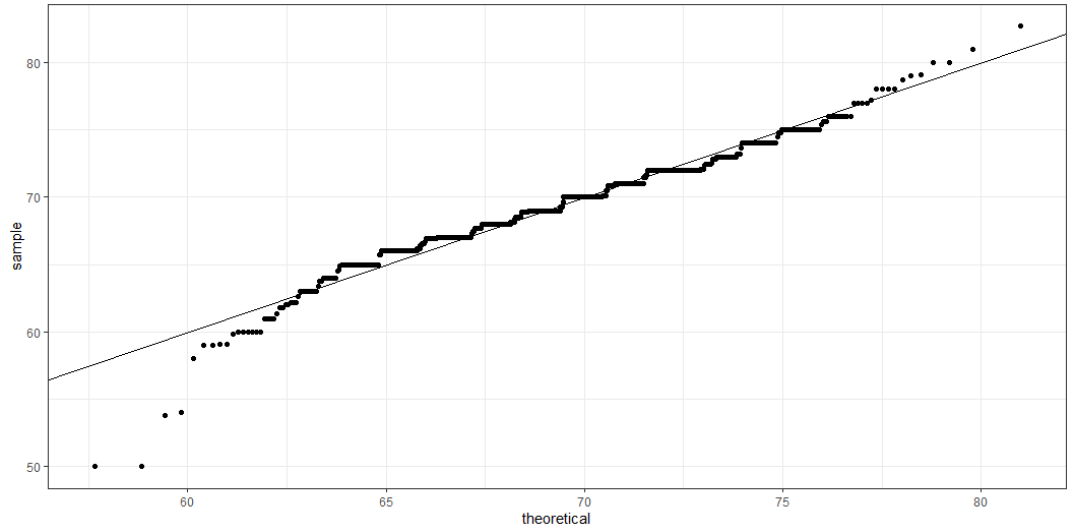
```
p + geom_qq(dparams = params)
```



We can add an identity line to see how well the normal approximation

works.

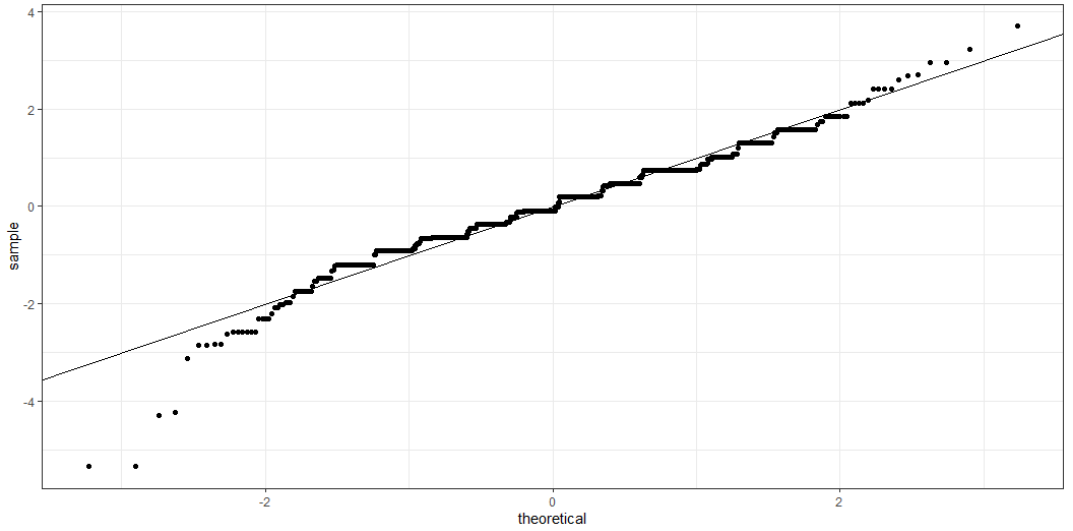
```
p + geom_qq(dparams = params) + geom_abline()
```



Q-Q plot

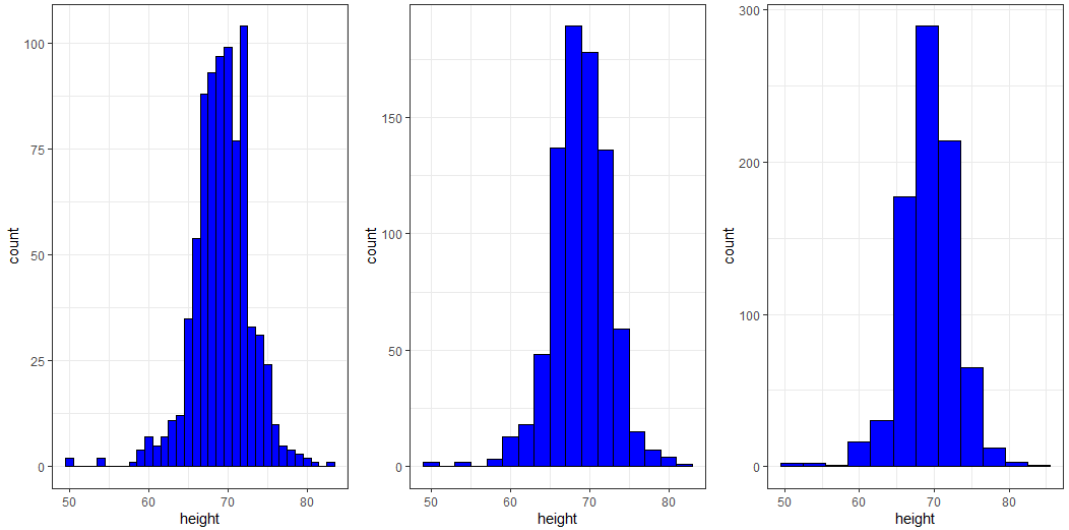
- We can also scale the data so that we have them in standard units and plot against the standard normal distribution.
- This will save us the step of having the compute the mean and standard deviation.

```
heights %>% filter(sex == "Male") %>%
ggplot(aes(sample = scale(height))) + geom_qq()
+ geom_abline()
```



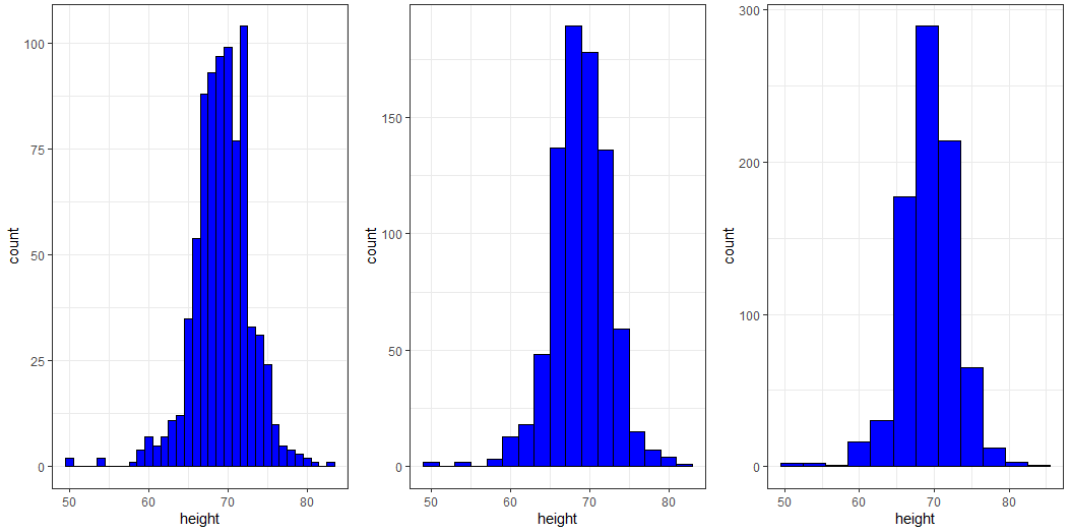
- One way to do this `gridExtra` package, and use its function `grid.arrange`.
- This lets us show different plot objects next to each other.

```
library(gridExtra)
p <- heights %>% filter(sex == "Male") %>% ggplot(aes(x
= height))
p1 <- p + geom_histogram(binwidth = 1, fill = "blue",
col = "black")
p2 <- p + geom_histogram(binwidth = 2, fill = "blue",
col = "black")
p3 <- p + geom_histogram(binwidth = 3, fill = "blue",
col = "black")
grid.arrange(p1, p2, p3, ncol = 3)
```



- One way to do this `gridExtra` package, and use its function `grid.arrange`.
- This lets us show different plot objects next to each other.

```
library(gridExtra)
p <- heights %>% filter(sex == "Male") %>% ggplot(aes(x
= height))
p1 <- p + geom_histogram(binwidth = 1, fill = "blue",
col = "black")
p2 <- p + geom_histogram(binwidth = 2, fill = "blue",
col = "black")
p3 <- p + geom_histogram(binwidth = 3, fill = "blue",
col = "black")
grid.arrange(p1, p2, p3, ncol = 3)
```



Summarizing Data with dplyr

- An important part of Exploratory Data Analysis is summarizing data.
- We learned about the average and standard deviation, two summary statistics for normally distributed data.
- We also learned better summaries can be achieved by splitting data into groups before using normal approximation. For example, male and female heights.

- `summarize`
- `group_by`
- `dot placeholder .$->` access resulting values
- `arrange` –> examine data after sorting

```
library(tidyverse)
library(dslabs)
data("heights")
# compute average and standard deviation for males
s <- heights %>% filter(sex == "Male") %>% summarize(average = mean(height), standard_deviation
= sd(height))
s
   average standard_deviation
1 69.31475           3.611024

# The resulting table stored in s is a data frame -> we can access the components with the
accessor dollar sign.
s$average
[1] 69.31475
s$standard_deviation
[1] 3.611024
```

- We can compute any summary that operates on vectors.

```
#compute median, minimum, maximum

heights %>%
  filter(sex == "Male") %>%
  summarize(median = median(height), minimum = min(height), maximum = max(height))
  median minimum  maximum
1     69      50 82.67717
```

- We can obtain there 3 values (median, min, max) using a single line code with `quantile` function.

```
heights %>% filter(sex == "Male") %>% reframe(range = quantile(height, c(0, 0.5, 1)))
     range
1 50.00000
2 69.00000
3 82.67717
```

- we will learn how to return vectors instead of data frames.

```
data("murders")
us_murder_rate <- murders %>% summarize(rate = sum(total) /
sum(population)*100000)
us_murder_rate
      rate
1 3.034555
class(us_murder_rate)
[1] "data.frame"
```

- Most of dplyr functions always return data frames.What if we need numeric value?

```
data("murders")
us_murder_rate <- murders %>% summarize(rate = sum(total) / sum(population)*100000)

us_murder_rate %>% .$rate
[1] 3.034555

us_murder_rate$rate
[1] 3.034555


us_murder_rate <- murders %>% summarize(rate = sum(total) / sum(population)*100000)
%>% .$rate

us_murder_rate
[1] 3.034555
```

- We split data to groups, then compute summaries for each group

```
heights %>% group_by(sex)
# A tibble: 1,050 × 2
# Groups:   sex [2]
   sex    height
   <fct>   <dbl>
 1 Male       75
 2 Male       70
 3 Male       68
 4 Male       74
 5 Male       61
 6 Female     65
 7 Female     66
 8 Female     62
 9 Female     66
10 Male       67
# ⓘ 1,040 more rows
```

- This is a special data frame called group data frame.
- dplyr functions, particularly summarize, will behave differently when acting on this object.
- Conceptually, you can think this object as many tables with the same columns but not necessarily the same rows that are stacked into one object.

Mean and Standard Deviation

```
heights %>% group_by(sex) %>% summarize(average = mean(height), standard_deviation = sd(height))
```

```
# A tibble: 2 × 3
  sex    average standard_deviation
  <fct>    <dbl>              <dbl>
1 Female    64.9               3.76
2 Male      69.3               3.61
```

Median murder rate in the four regions of the country

```
murders <- murders %>% mutate(murder_rate = total / population*100000)
murders %>% group_by(region) %>% summarize(median_rate = median(murder_rate))
```

```
# A tibble: 4 × 2
  region        median_rate
  <fct>               <dbl>
1 Northeast            1.80
2 South                3.40
3 North Central        1.97
4 West                 1.29
```

- sort table by different columns.
- we already know about the order and sort functions. dplyr has useful `arrange`

- **Order states by their population size:**

```
murders %>% arrange(population) %>% head()
```

	state	abb	region	population	total	murder_rate
1	Wyoming	WY	West	563626	5	0.8871131
2	District of Columbia	DC	South	601723	99	16.4527532
3	Vermont	VT	Northeast	625741	2	0.3196211
4	North Dakota	ND	North Central	672591	4	0.5947151
5	Alaska	AK	West	710231	19	2.6751860
6	South Dakota	SD	North Central	814180	8	0.9825837

Order states by murder_rate

```
murders %>% arrange(murder_rate) %>% head()
```

```
          state abb        region population total murder_rate
1       Vermont  VT     Northeast     625741     2   0.3196211
2 New Hampshire  NH     Northeast    1316470     5   0.3798036
3        Hawaii  HI          West    1360301     7   0.5145920
4  North Dakota  ND North Central     672591     4   0.5947151
5          Iowa  IA North Central    3046355    21   0.6893484
6         Idaho  ID          West    1567582    12   0.7655102
```

To make it descending instead of ascending

```
murders %>% arrange(desc(murder_rate)) %>% head()
```

```
                  state abb        region population total murder_rate
1 District of Columbia  DC         South     601723    99   16.452753
2             Louisiana  LA         South    4533372   351    7.742581
3              Missouri  MO North Central    5988927   321    5.359892
4              Maryland  MD         South    5773552   293    5.074866
5        South Carolina  SC         South    4625364   207    4.475323
6              Delaware  DE         South     897934    38    4.231937
```

Nested sorting

```
murders %>% arrange(region, murder_rate) %>% head()
          state abb    region population total murder_rate
1       Vermont  VT Northeast     625741     2   0.3196211
2 New Hampshire  NH Northeast    1316470     5   0.3798036
3         Maine  ME Northeast    1328361    11   0.8280881
4  Rhode Island  RI Northeast    1052567    16   1.5200933
5 Massachusetts  MA Northeast    6547629   118   1.8021791
6      New York  NY Northeast   19378102   517   2.6679599
```

top_n

- we used head() function to see first 6 rows. if we want to see a larger proportion of the data, say the to 10:

```
murders %>% top_n(10, murder_rate)
```

```
                  state abb        region population total murder_rate
1               Arizona  AZ          West    6392017   232    3.629527
2              Delaware  DE         South     897934    38    4.231937
3  District of Columbia  DC         South     601723    99   16.452753
4               Georgia  GA         South    9920000   376    3.790323
5             Louisiana  LA         South    4533372   351    7.742581
6              Maryland  MD         South    5773552   293    5.074866
7              Michigan  MI North Central    9883640   413    4.178622
8           Mississippi  MS         South    2967297   120    4.044085
9              Missouri  MO North Central    5988927   321    5.359892
10       South Carolina  SC         South    4625364   207    4.475323
```

top_n

To make them ordered, we can use arrange function:

```
murders %>% arrange(desc(murder_rate)) %>% top_n(10)
                  state abb        region population total murder_rate
1  District of Columbia  DC         South     601723    99   16.452753
2             Louisiana  LA         South    4533372   351    7.742581
3              Missouri  MO North Central    5988927   321    5.359892
4              Maryland  MD         South    5773552   293    5.074866
5        South Carolina  SC         South    4625364   207    4.475323
6              Delaware  DE         South     897934    38    4.231937
7              Michigan  MI North Central    9883640   413    4.178622
8           Mississippi  MS         South    2967297   120    4.044085
9               Georgia  GA         South    9920000   376    3.790323
10              Arizona  AZ          West    6392017   232    3.629527
```


Any
Questions