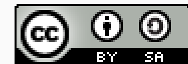
Section A01 **CDS 101** *Introduction to Computational and Data Sciences*

Class 9: Statistical distributions II

June 1, 2018



These slides are licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.

General

Announcements

- Homework 2 posted, due datae is June 6th @ 11:59pm: http://summer18.cds101.com/assignments/homework-2/
- Reading 9 from R for Data Science, questions due on June 6th by 9:00am
    - All of chapter 7

Statistical distributions

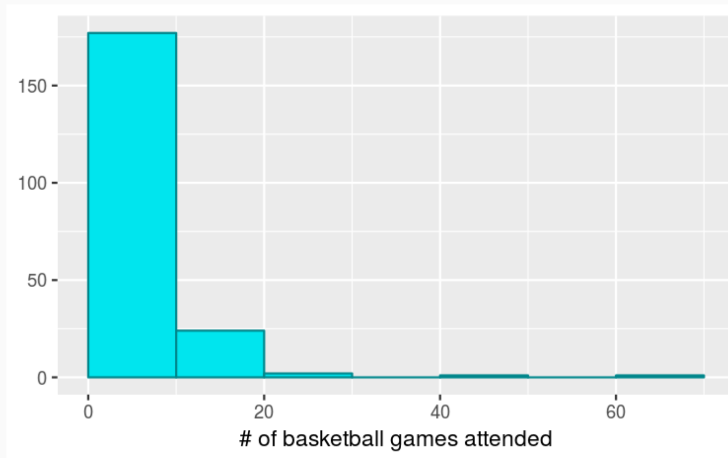
Extremely skewed data

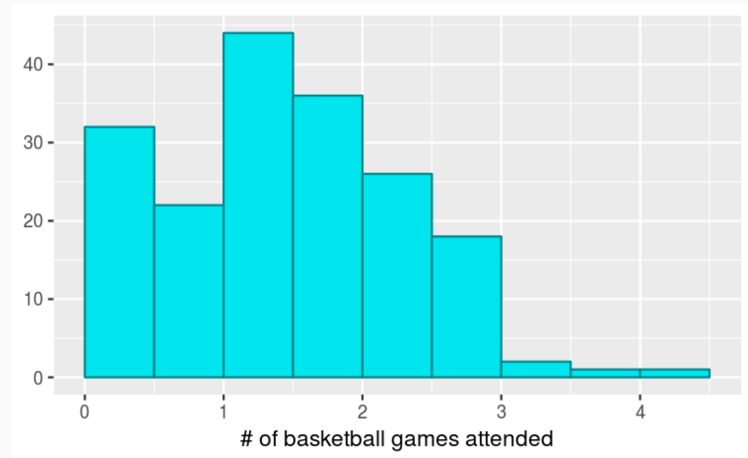
When data are extremely skewed, transforming them might make modeling easier. A common transformation is the **log transformation**.

Extremely skewed data

When data are extremely skewed, transforming them might make modeling easier. A common transformation is the **log transformation**.

The histograms on the left shows the distribution of number of basketball games attended by students. The histogram on the right shows the distribution of log of number of games attended.





Pros and cons of transformations

- Skewed data are easier to model with when they are transformed because outliers tend to become far less prominent after an appropriate transformation.

```
# of games             70      50      25     ···
log10(# of games)    4.25    3.91    3.22     ···
```

- However, results of an analysis might be difficult to interpret because the log of a measured variable is usually meaningless.

Pros and cons of transformations

- Skewed data are easier to model with when they are transformed because outliers tend to become far less prominent after an appropriate transformation.

```
# of games             70      50      25    ...
log10(# of games)    4.25    3.91    3.22    ...
```

- However, results of an analysis might be difficult to interpret because the log of a measured variable is usually meaningless.

What other variables would you expect to be extremely skewed?

Pros and cons of transformations

- Skewed data are easier to model with when they are transformed because outliers tend to become far less prominent after an appropriate transformation.

```
# of games             70     50     25    ···
log10(# of games)    4.25   3.91   3.22    ···
```

- However, results of an analysis might be difficult to interpret because the log of a measured variable is usually meaningless.

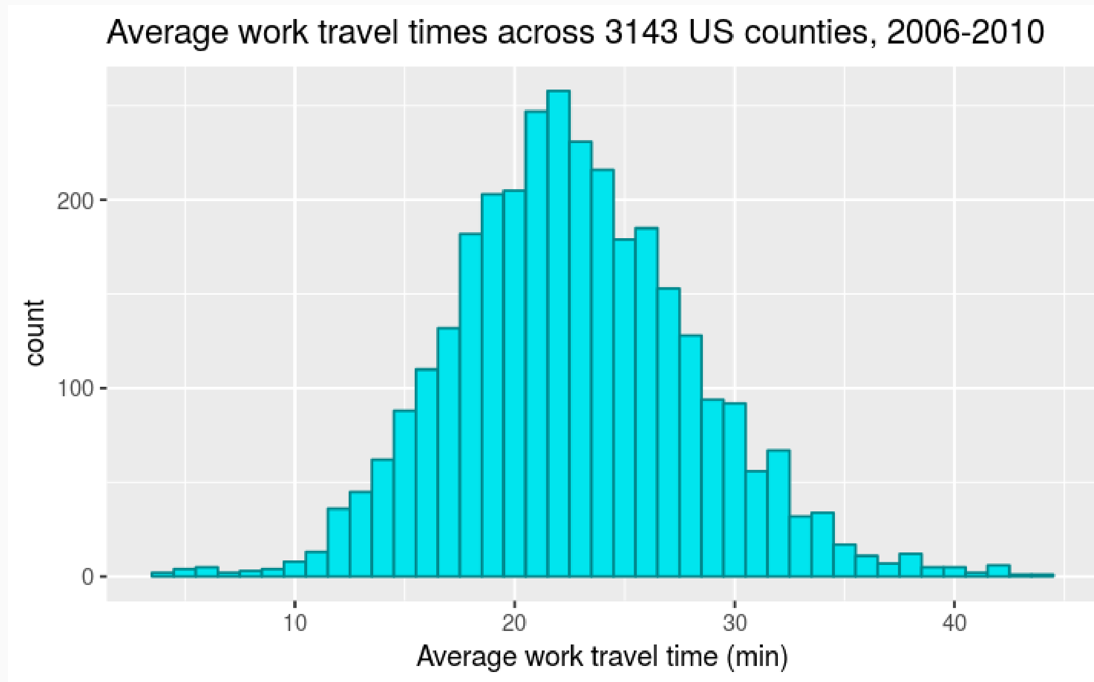
What other variables would you expect to be extremely skewed?

Salary, housing prices, etc.

Quantifying statistical distributions in R

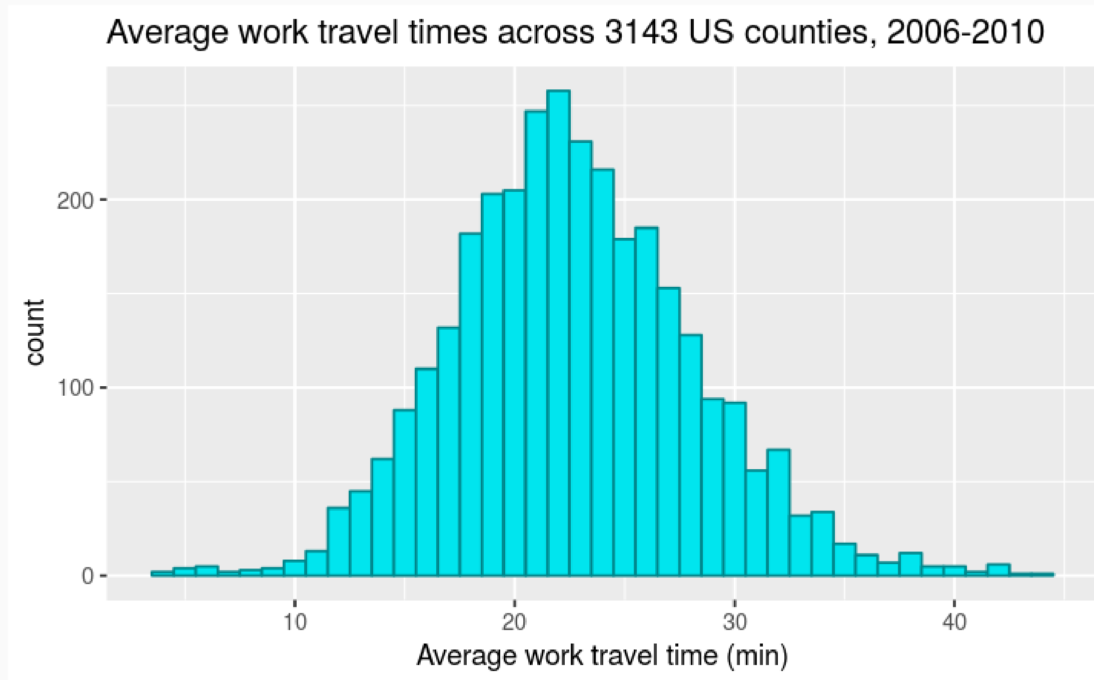
Example data distribution

The following distribution comes from data posted by the US Census Bureau:



Example data distribution

The following distribution comes from data posted by the US Census Bureau:



How can we quantify the shape of this distribution?

Useful statistical functions

The following R functions will be useful for computing basic statistical measures of any numerical data column (variable)

Useful statistical functions

The following R functions will be useful for computing basic statistical measures of any numerical data column (variable)

- `mean()`: Computes the average

Useful statistical functions

The following R functions will be useful for computing basic statistical measures of any numerical data column (variable)

- `mean()`: Computes the average
- `median()`: Computes the median

Useful statistical functions

The following R functions will be useful for computing basic statistical measures of any numerical data column (variable)

- `mean()`: Computes the average
- `median()`: Computes the median
- `min()`: Finds the minimum value

Useful statistical functions

The following R functions will be useful for computing basic statistical measures of any numerical data column (variable)

- `mean()`: Computes the average
- `median()`: Computes the median
- `min()`: Finds the minimum value
- `max()`: Finds the maximum value

Useful statistical functions

The following R functions will be useful for computing basic statistical measures of any numerical data column (variable)

- `mean()`: Computes the average
- `median()`: Computes the median
- `min()`: Finds the minimum value
- `max()`: Finds the maximum value
- `sd()`: Computes the standard deviation

Useful statistical functions

The following R functions will be useful for computing basic statistical measures of any numerical data column (variable)

- `mean()`: Computes the average
- `median()`: Computes the median
- `min()`: Finds the minimum value
- `max()`: Finds the maximum value
- `sd()`: Computes the standard deviation
- `IQR()`: Computes the interquartile range

Useful statistical functions

The following R functions will be useful for computing basic statistical measures of any numerical data column (variable)

- `mean()`: Computes the average
- `median()`: Computes the median
- `min()`: Finds the minimum value
- `max()`: Finds the maximum value
- `sd()`: Computes the standard deviation
- `IQR()`: Computes the interquartile range
- `percent_rank()`: Computes percentiles

Using the statistical functions

Using the statistical functions

- Every function except `percent_rank()` will always return a single quantity

Using the statistical functions

- Every function except `percent_rank()` will always return a single quantity
- The `summarize()` function is appropriate here:

Using the statistical functions

- Every function except `percent_rank()` will always return a single quantity
- The `summarize()` function is appropriate here:

```
county %>%
  summarize(
    mean = mean(mean_work_travel),
    median = median(mean_work_travel),
    min = min(mean_work_travel),
    max = max(mean_work_travel),
    sd = sd(mean_work_travel),
    iqr = IQR(mean_work_travel)
  )
```

Using the statistical functions

- Every function except `percent_rank()` will always return a single quantity
- The `summarize()` function is appropriate here:

```
county %>%
  summarize(
    mean = mean(mean_work_travel),
    median = median(mean_work_travel),
    min = min(mean_work_travel),
    max = max(mean_work_travel),
    sd = sd(mean_work_travel),
    iqr = IQR(mean_work_travel)
  )
```

mean	median	min	max	sd	iqr
22.72558	22.4	4.3	44.2	5.514159	7.1

Using the statistical functions

- `percent_rank()` operates on the full column of values, so it needs to be paired with `mutate()`

Using the statistical functions

- `percent_rank()` operates on the full column of values, so it needs to be paired with `mutate()`
- Once we have the percentiles, we can find the cutoff value for each percentile

Using the statistical functions

- `percent_rank()` operates on the full column of values, so it needs to be paired with `mutate()`
- Once we have the percentiles, we can find the cutoff value for each percentile

```
county %>%
  mutate(
    percentile = percent_rank(mean_work_travel),
    quartile = case_when(                      # case_when() similar to if_else()
      percentile < 0.25 ~ "Q1",                # label between 0 and 0.25 as Q1,
      between(percentile, 0.25, 0.50) ~ "Q2",  # between 0.25 and 0.50 as Q2,
      between(percentile, 0.50, 0.75) ~ "Q3",  # between 0.50 and 0.75 as Q3,
      percentile >= 0.75 ~ "Q4"                # and 0.75 to 1.00 as Q4
    )
  ) %>%
  group_by(quartile) %>%
  summarize(cutoff = max(mean_work_travel))    # cutoff is maximum in quartile
```

Using the statistical functions

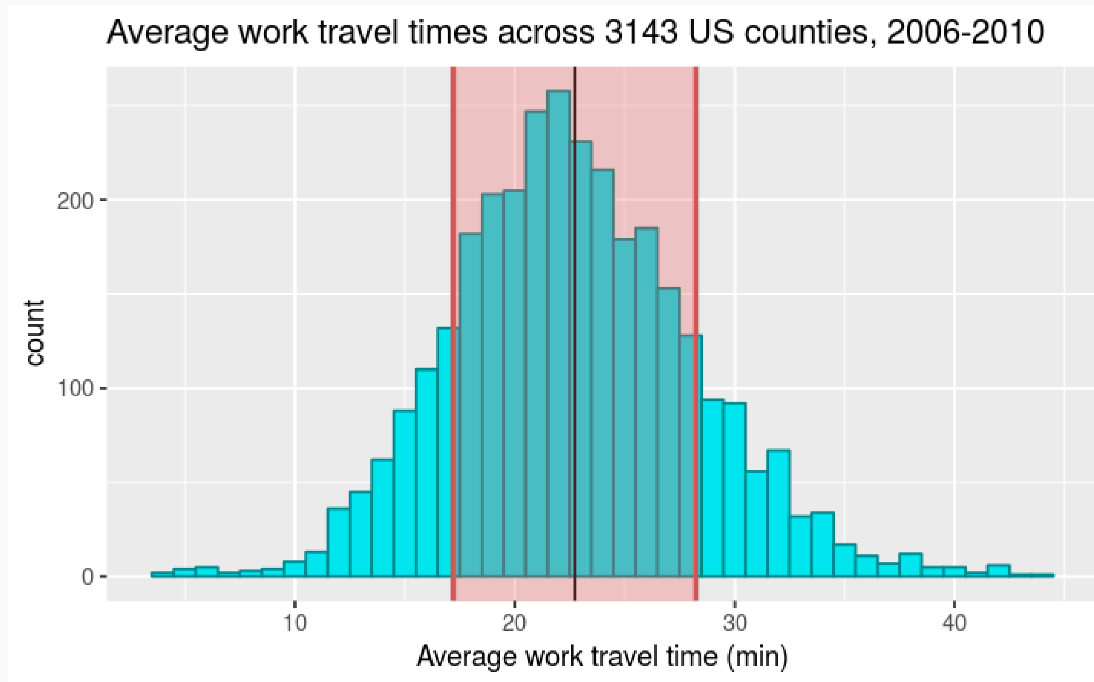
- `percent_rank()` operates on the full column of values, so it needs to be paired with `mutate()`
- Once we have the percentiles, we can find the cutoff value for each percentile

```
county %>%
  mutate(
    percentile = percent_rank(mean_work_travel),
    quartile = case_when(                       # case_when() similar to if_else()
      percentile < 0.25 ~ "Q1",                 # label between 0 and 0.25 as Q1,
      between(percentile, 0.25, 0.50) ~ "Q2",   # between 0.25 and 0.50 as Q2,
      between(percentile, 0.50, 0.75) ~ "Q3",   # between 0.50 and 0.75 as Q3,
      percentile >= 0.75 ~ "Q4"                 # and 0.75 to 1.00 as Q4
    )
  ) %>%
  group_by(quartile) %>%
  summarize(cutoff = max(mean_work_travel))    # cutoff is maximum in quartile
```

Q1	Q2	Q3	Q4
19	22.4	26.1	44.2

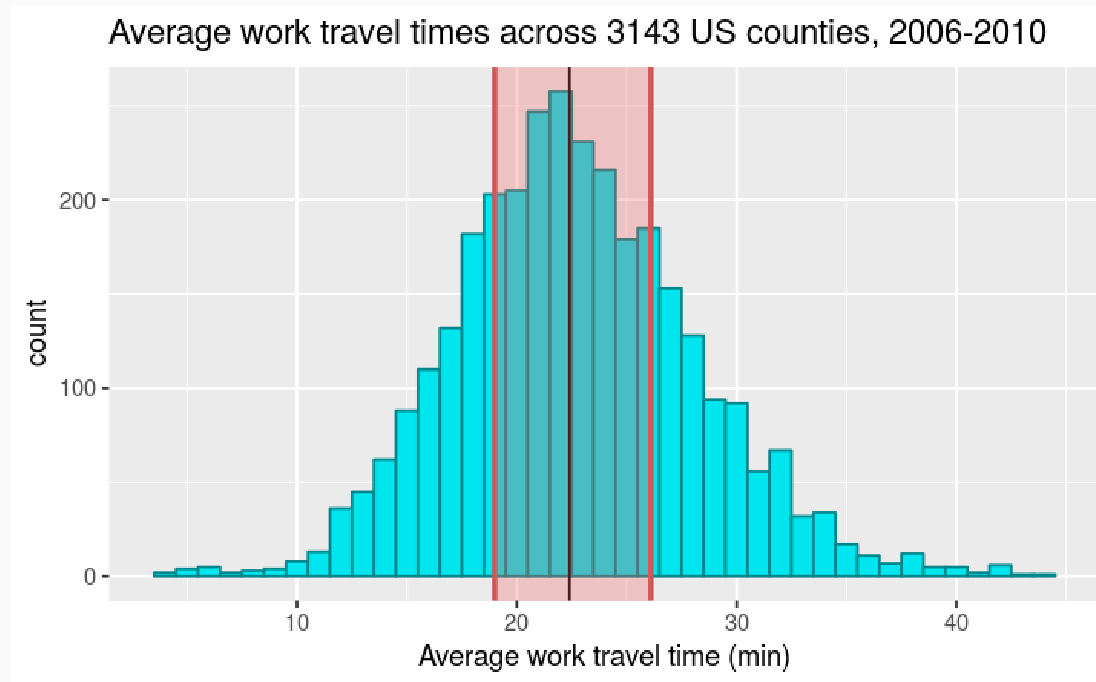
Interpreting summary statistics: mean, sd

One standard deviation above and below the mean



Interpreting summary statistics: median, IQR

The median and inter-quartile range



From histograms to probability mass functions

Data distributions

- We've already learned that histograms (`geom_histogram()`) are a convenient way to represent numerical data in a single column (variable)

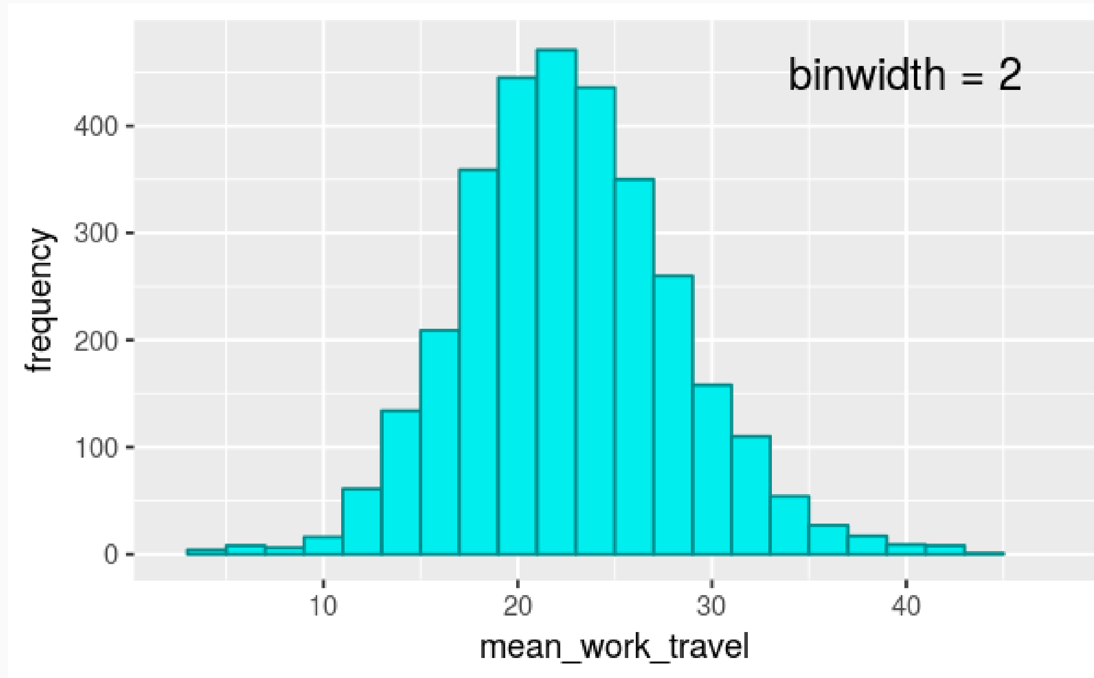
Data distributions

- We've already learned that histograms (`geom_histogram()`) are a convenient way to represent numerical data in a single column (variable)

mean_work_travel
25.1
25.8
23.8
28.3
33.2
28.1
25.1
...

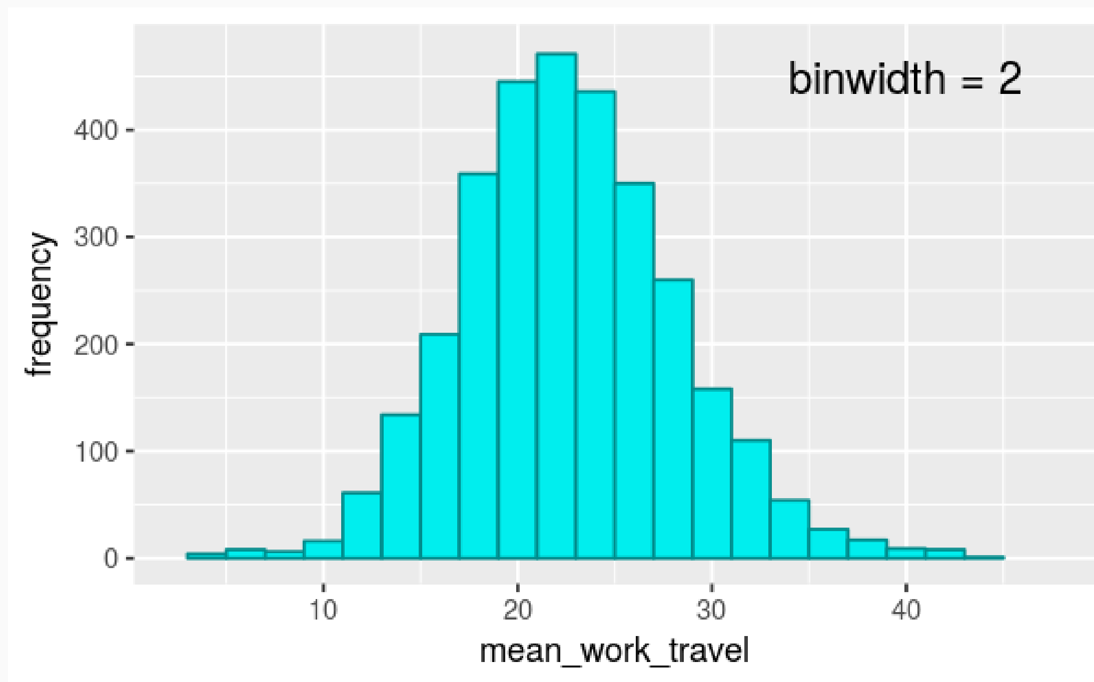
Data distributions

- We've already learned that histograms (`geom_histogram()`) are a convenient way to represent numerical data in a single column (variable)



Data distributions

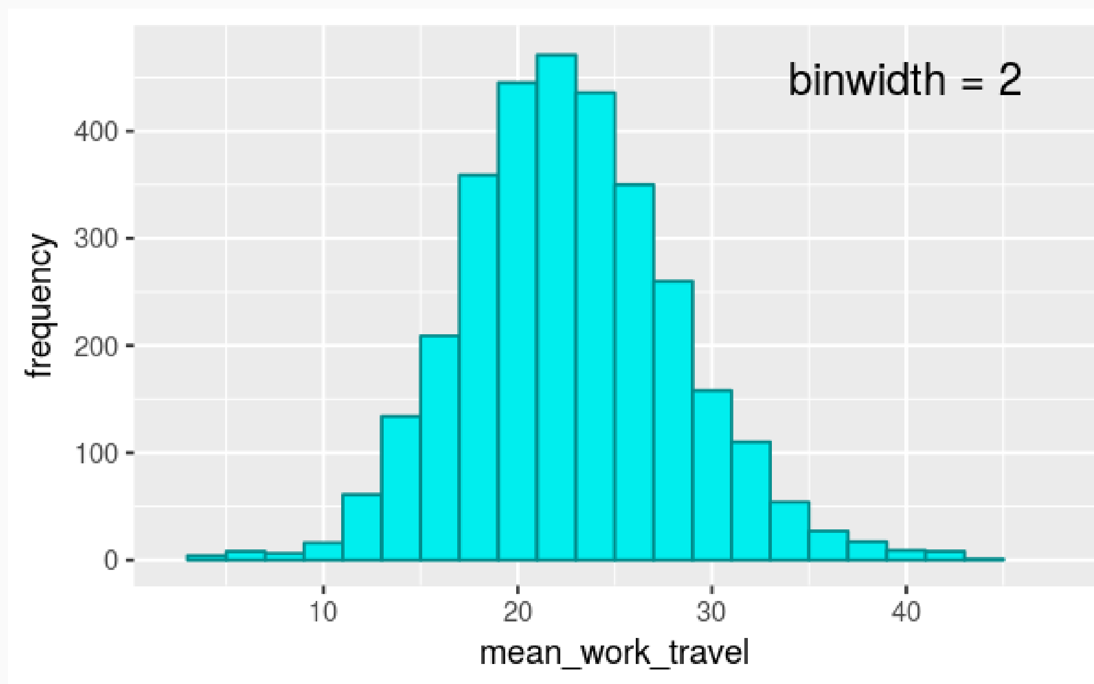
- We've already learned that histograms (`geom_histogram()`) are a convenient way to represent numerical data in a single column (variable)



- A histogram represents the **frequency** that values show up for a given variable

Data distributions

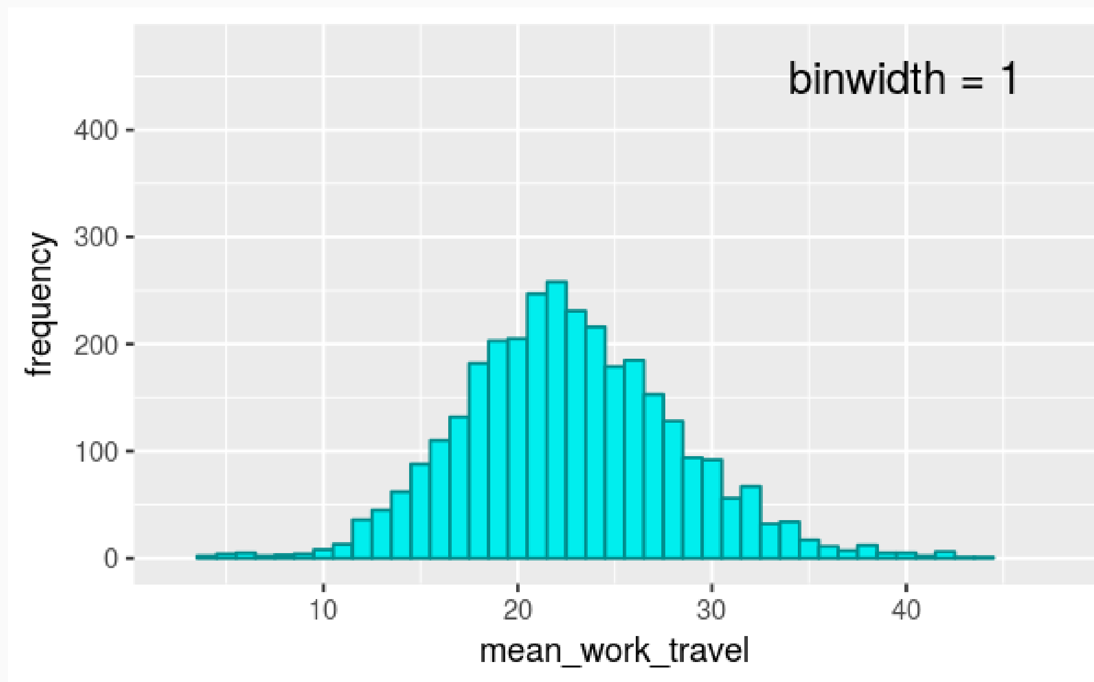
- We've already learned that histograms (`geom_histogram()`) are a convenient way to represent numerical data in a single column (variable)



- A histogram represents the **frequency** that values show up for a given variable
- `binwidth` changes the "buckets" for the data, impacting the frequency heights.

Data distributions

- We've already learned that histograms (`geom_histogram()`) are a convenient way to represent numerical data in a single column (variable)



- A histogram represents the **frequency** that values show up for a given variable
- `binwidth` changes the "buckets" for the data, impacting the frequency heights

Comparing distributions with unequal observations

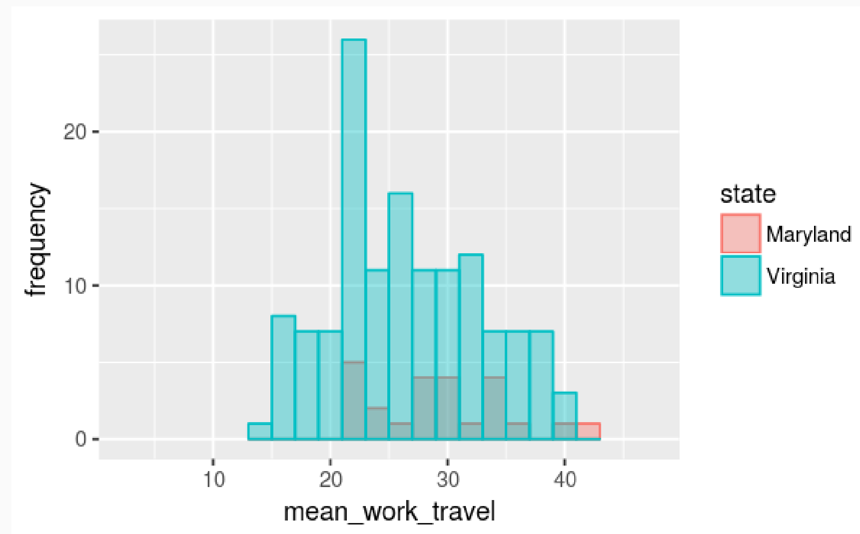
- So far, we've largely skipped over the question of how to compare distributions with varying numbers of observations

Comparing distributions with unequal observations

- So far, we've largely skipped over the question of how to compare distributions with varying numbers of observations
- In our current example of average times to travel to work, we can group the data by state and compare Virginia to Maryland

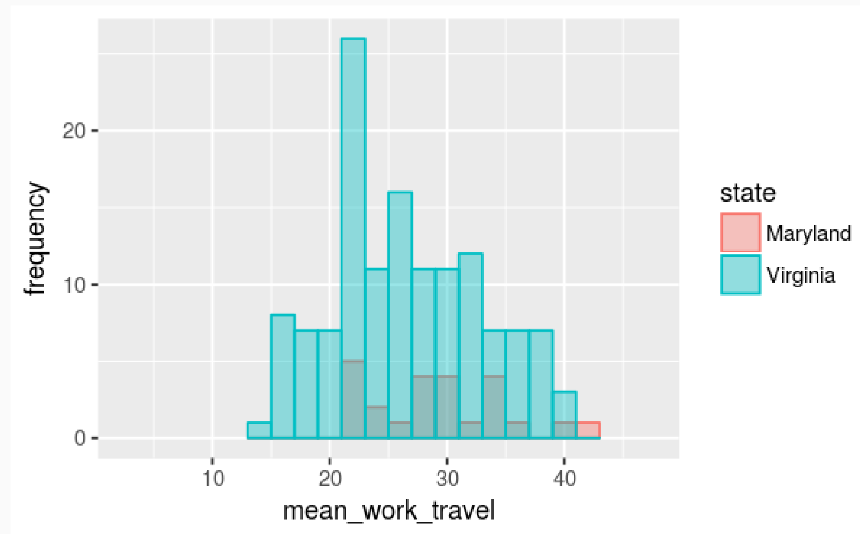
Comparing distributions with unequal observations

- So far, we've largely skipped over the question of how to compare distributions with varying numbers of observations
- In our current example of average times to travel to work, we can group the data by state and compare Virginia to Maryland



Comparing distributions with unequal observations

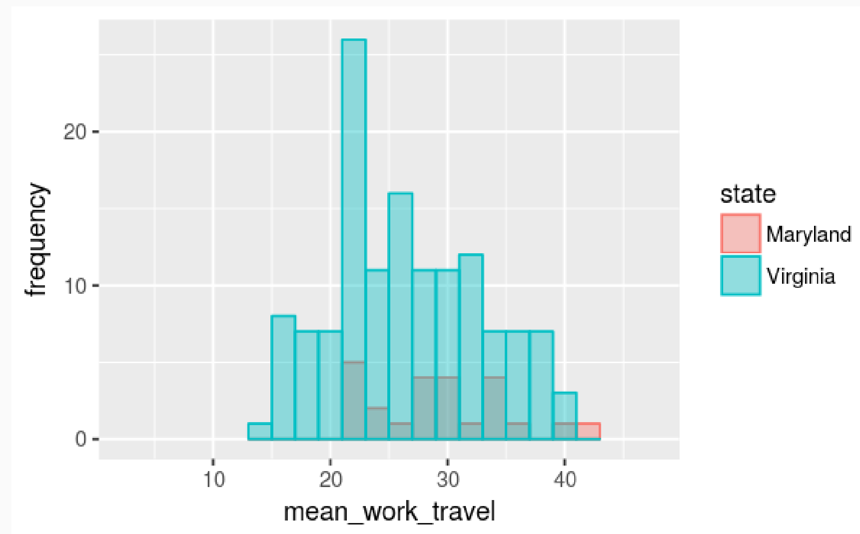
- So far, we've largely skipped over the question of how to compare distributions with varying numbers of observations
- In our current example of average times to travel to work, we can group the data by state and compare Virginia to Maryland



In which state am I more likely to have a 30 minute commute?

Comparing distributions with unequal observations

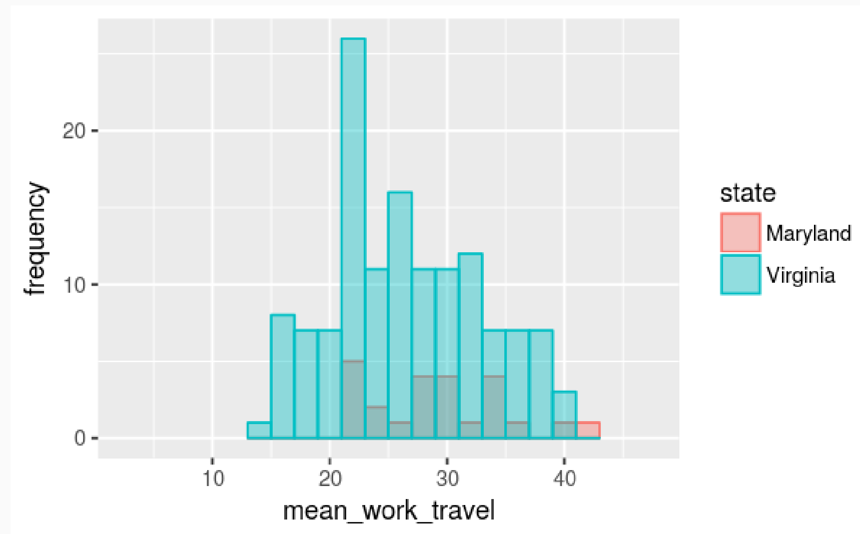
- So far, we've largely skipped over the question of how to compare distributions with varying numbers of observations
- In our current example of average times to travel to work, we can group the data by state and compare Virginia to Maryland



Comparing distributions with unequal observations

- So far, we've largely skipped over the question of how to compare distributions with varying numbers of observations

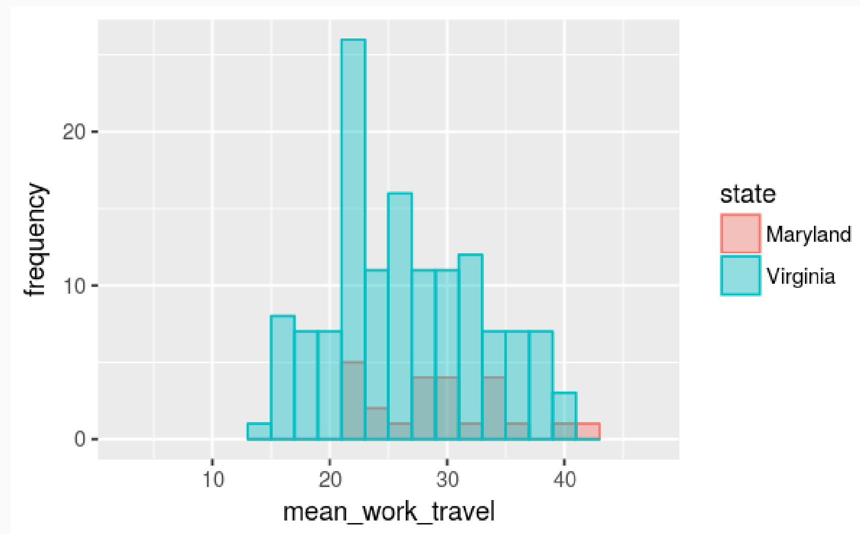
- In our current example of average times to travel to work, we can group the data by state and compare Virginia to Maryland



- In the dataset, Virginia has 134 counties compared to Maryland's 24 counties

Comparing distributions with unequal observations

- So far, we've largely skipped over the question of how to compare distributions with varying numbers of observations
- In our current example of average times to travel to work, we can group the data by state and compare Virginia to Maryland



- In the dataset, Virginia has 134 counties compared to Maryland's 24 counties
- We need to **normalize** the frequency counts

From frequency to probability

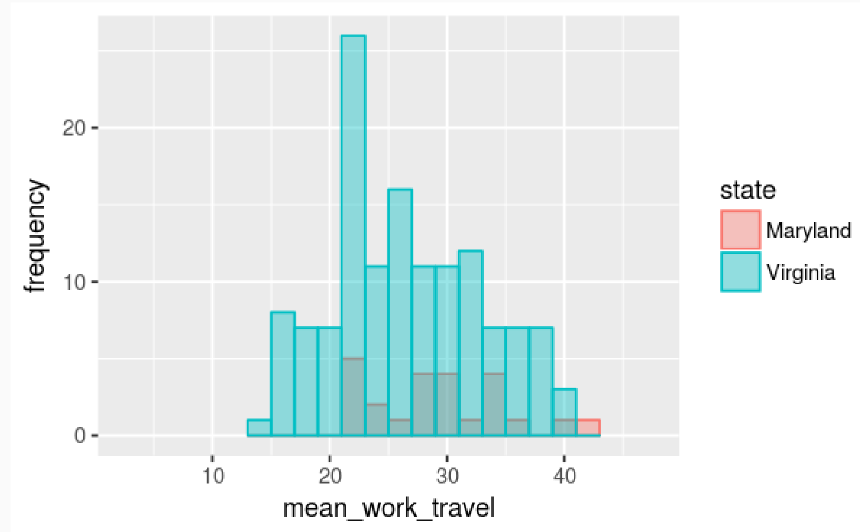
- Normalization is straightforward, just divide the frequency count in each "bucket" by the total number of observations in the histogram

From frequency to probability

- Normalization is straightforward, just divide the frequency count in each "bucket" by the total number of observations in the histogram
- If you group by categories, that you should divide by the number of observations in each group

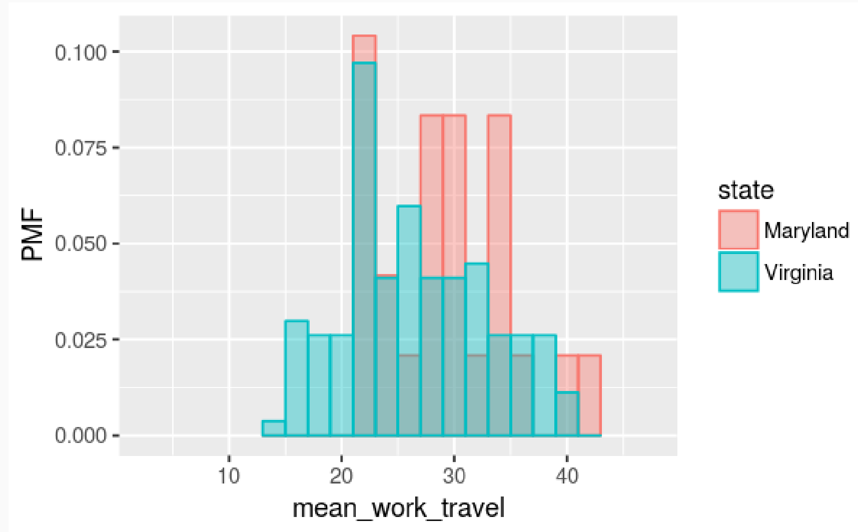
From frequency to probability

- Normalization is straightforward, just divide the frequency count in each "bucket" by the total number of observations in the histogram
- If you group by categories, that you should divide by the number of observations in each group
- To normalize the histograms from the prior example, we need to divide the Virginia frequencies by 134 and the Maryland frequencies by 24

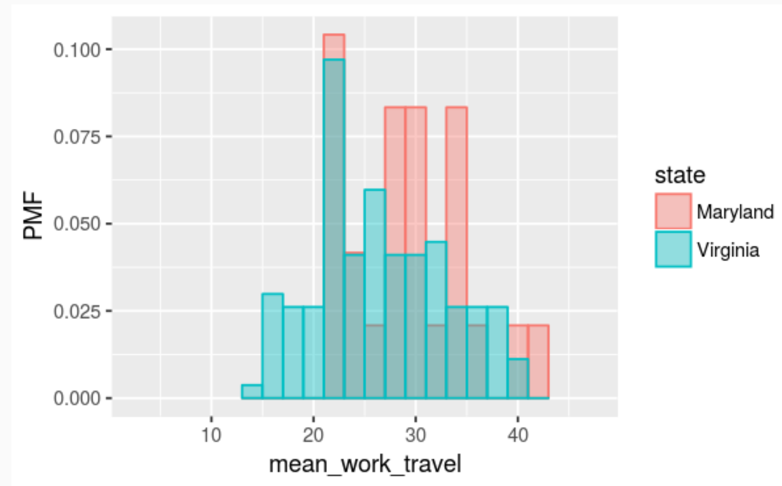


From frequency to probability

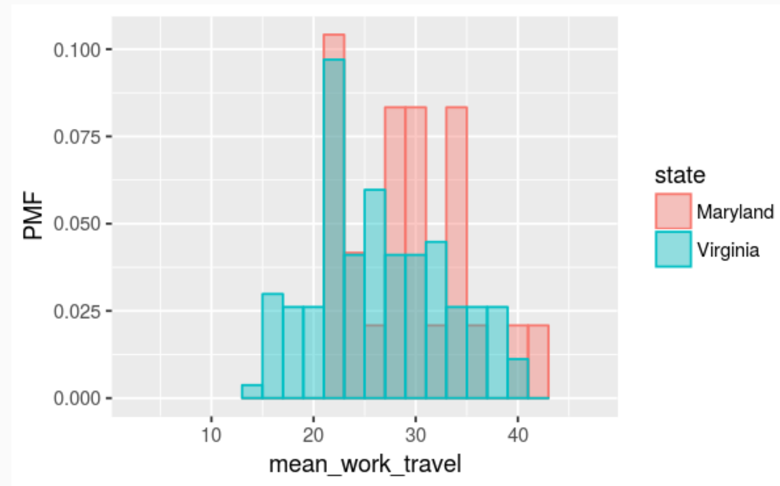
- Normalization is straightforward, just divide the frequency count in each "bucket" by the total number of observations in the histogram
- If you group by categories, that you should divide by the number of observations in each group
- To normalize the histograms from the prior example, we need to divide the Virginia frequencies by 134 and the Maryland frequencies by 24



Probability mass function (PMF)

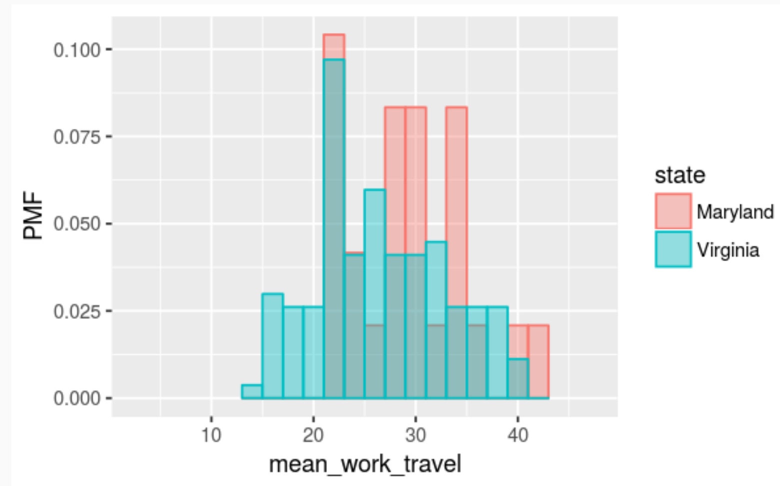


Probability mass function (PMF)



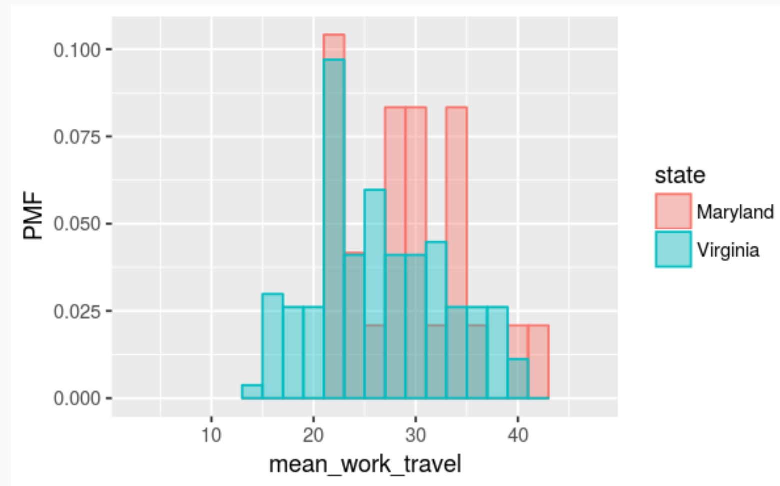
- Just like a histogram, except that the bar heights reflect **probabilities** instead of **frequency counts**

Probability mass function (PMF)



- Just like a histogram, except that the bar heights reflect **probabilities** instead of **frequency counts**
- Allows for a meaningful comparison of distributions with different numbers of observations

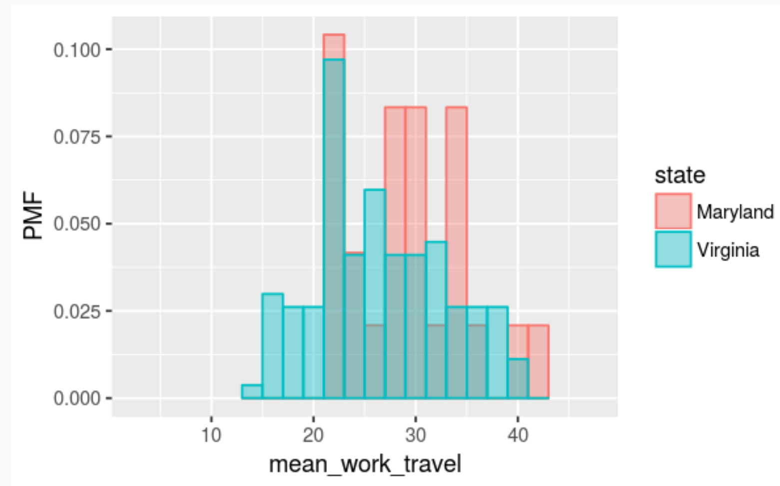
Probability mass function (PMF)



- Just like a histogram, except that the bar heights reflect **probabilities** instead of **frequency counts**
- Allows for a meaningful comparison of distributions with different numbers of observations

In which state am I more likely to have a 30 minute commute?

Probability mass function (PMF)



- Just like a histogram, except that the bar heights reflect **probabilities** instead of **frequency counts**
- Allows for a meaningful comparison of distributions with different numbers of observations

In which state am I more likely to have a 30 minute commute?

Maryland

Creating PMFs in R

- With `ggplot2`, it's straightforward to convert a histogram into a PMF

Creating PMFs in R

- With `ggplot2`, it's straightforward to convert a histogram into a PMF

```
county %>%
  filter(state == "Virginia" | state == "Maryland") %>%
  ggplot() +
  geom_histogram(
    mapping = aes(x = mean_work_travel, fill = state),
    position = "identity",
    alpha = 0.5
  )
```

Creating PMFs in R

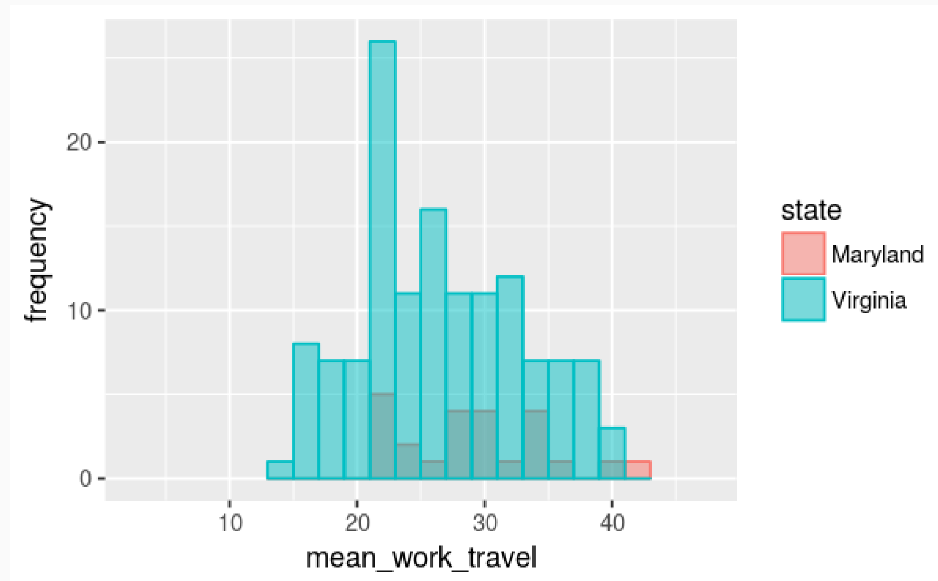
- With `ggplot2`, it's straightforward to convert a histogram into a PMF

```
county %>%
  filter(state == "Virginia" | state == "Maryland") %>%
  ggplot() +
  geom_histogram(
    mapping = aes(x = mean_work_travel, fill = state),
    position = "identity",
    alpha = 0.5
  )
```

Creating PMFs in R

- With `ggplot2`, it's straightforward to convert a histogram into a PMF

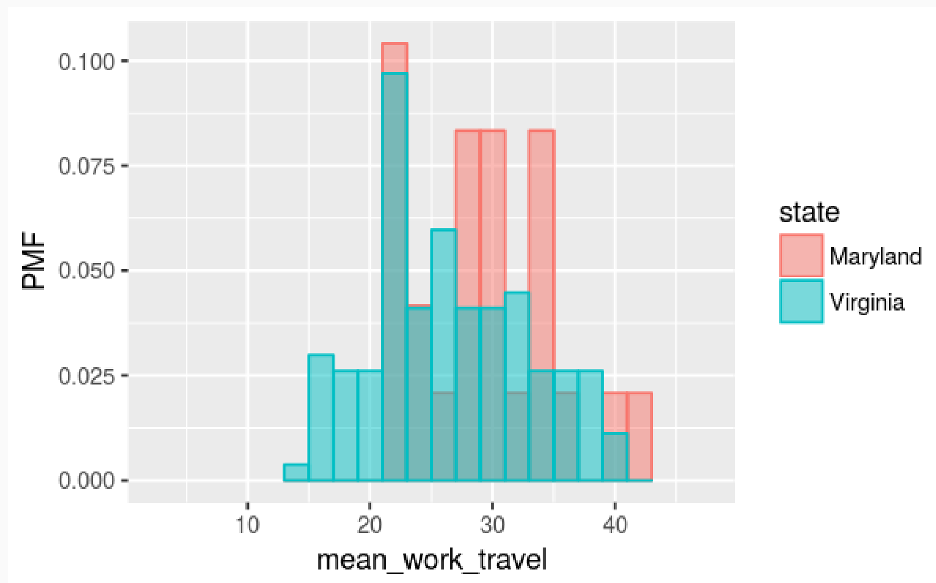
```
county %>%
  filter(state == "Virginia" | state == "Maryland") %>%
  ggplot() +
  geom_histogram(
    mapping = aes(x = mean_work_travel, fill = state),
    position = "identity",
    alpha = 0.5
  )
```



Creating PMFs in R

- With `ggplot2`, it's straightforward to convert a histogram into a PMF

```
county %>%
  filter(state == "Virginia" | state == "Maryland") %>%
  ggplot() +
  geom_histogram(
    mapping = aes(x = mean_work_travel, y = ..density.., fill = state),
    position = "identity",
    alpha = 0.5
  )
```



Obtaining PMF values

Obtaining PMF values

1. Compute them manually

Obtaining PMF values

1. Compute them manually
2. Extract them from your `ggplot2` visualization

Obtaining PMF values

1. Compute them manually

2. Extract them from your `ggplot2` visualization

Obtaining PMF values

1. Compute them manually

2. Extract them from your `ggplot2` visualization

Assign the figure to a variable

```
va_md_pmf_figure <- county %>%
  filter(state == "Virginia" | state == "Maryland") %>%
  ggplot() +
  geom_histogram(
    mapping = aes(x = mean_work_travel, y = ..density.., fill = state),
    binwidth = 2,
    center = 0
  )
```

Obtaining PMF values

1. Compute them manually

2. Extract them from your `ggplot2` visualization

Assign the figure to a variable

```
va_md_pmf_figure <- county %>%
  filter(state == "Virginia" | state == "Maryland") %>%
  ggplot() +
  geom_histogram(
    mapping = aes(x = mean_work_travel, y = ..density.., fill = state),
    binwidth = 2,
    center = 0
  )
```

Use `ggplot_build()` with `pluck()` and `as_tibble()` as follows:

```
va_md_pmf_data <- va_md_pmf_figure %>%
  ggplot_build() %>%
  pluck("data", 1) %>%
  as_data_frame()
```

Obtaining PMF values

```
va_md_pmf_data %>%
  glimpse()
```

```
## Observations: 30
## Variables: 17
## $ fill     <chr> "#00BFC4", "#F8766D", "#00BFC4", "#F8766D", "#00BFC4"...
## $ y        <dbl> 0.003731343, 0.003731343, 0.029850746, 0.029850746, 0...
## $ count    <dbl> 1, 0, 8, 0, 7, 0, 7, 0, 26, 5, 11, 2, 16, 1, 11, 4, 1...
## $ x        <dbl> 14, 14, 16, 16, 18, 18, 20, 20, 22, 22, 24, 24, 26, 2...
## $ xmin     <dbl> 13, 13, 15, 15, 17, 17, 19, 19, 21, 21, 23, 23, 25, 2...
## $ xmax     <dbl> 15, 15, 17, 17, 19, 19, 21, 21, 23, 23, 25, 25, 27, 2...
## $ density  <dbl> 0.003731343, 0.000000000, 0.029850746, 0.000000000, 0...
## $ ncount   <dbl> 0.03846154, 0.00000000, 0.30769231, 0.00000000, 0.269...
## $ ndensity <dbl> 10.30769, 0.00000, 82.46154, 0.00000, 72.15385, 0.000...
## $ PANEL    <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,...
## $ group    <int> 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1,...
## $ ymin     <dbl> 0.000000000, 0.003731343, 0.000000000, 0.029850746, 0...
## $ ymax     <dbl> 0.003731343, 0.003731343, 0.029850746, 0.029850746, 0...
## $ colour   <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
## $ size     <dbl> 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5...
## $ linetype <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,...
## $ alpha    <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
```

Obtaining PMF values

To get the Maryland PMF data:

```
md_pmf_data <- va_md_pmf_data %>%
  filter(group == 1) %>%
  select(x, density)
```

x	density
14	0
16	0
18	0
20	0
22	0.104166666666667
24	0.0416666666666667
26	0.0208333333333333
…	…

Obtaining PMF values

To get the Virginia PMF data:

```
va_pmf_data <- va_md_pmf_data %>%
  filter(group == 2) %>%
  select(x, density)
```

x	density
14	0.00373134328358209
16	0.0298507462686567
18	0.0261194029850746
20	0.0261194029850746
22	0.0970149253731343
24	0.041044776119403
26	0.0597014925373134
…	…

Cumulative distribution functions

Data by percentile rank

Data by percentile rank

- PMFs are handy exploratory tools, but as with histograms, the binwidth can strongly influence what your plot looks like

Data by percentile rank

- PMFs are handy exploratory tools, but as with histograms, the binwidth can strongly influence what your plot looks like
- We can overcome this problem if we convert the data into a sorted list of percentile ranks

Data by percentile rank

- PMFs are handy exploratory tools, but as with histograms, the binwidth can strongly influence what your plot looks like
- We can overcome this problem if we convert the data into a sorted list of percentile ranks
- **Advantages**

Data by percentile rank

- PMFs are handy exploratory tools, but as with histograms, the binwidth can strongly influence what your plot looks like
- We can overcome this problem if we convert the data into a sorted list of percentile ranks
- **Advantages**
  - Don't need to select a binsize

Data by percentile rank

- PMFs are handy exploratory tools, but as with histograms, the binwidth can strongly influence what your plot looks like
- We can overcome this problem if we convert the data into a sorted list of percentile ranks
- **Advantages**
  - Don't need to select a binsize
  - Easier to compare similarities and differences of different data distributions

Data by percentile rank

- PMFs are handy exploratory tools, but as with histograms, the binwidth can strongly influence what your plot looks like
- We can overcome this problem if we convert the data into a sorted list of percentile ranks
- **Advantages**
  - Don't need to select a binsize
  - Easier to compare similarities and differences of different data distributions
  - Different classes of data distributions have distinct shapes

Data by percentile rank

- PMFs are handy exploratory tools, but as with histograms, the binwidth can strongly influence what your plot looks like
- We can overcome this problem if we convert the data into a sorted list of percentile ranks
- **Advantages**
  - Don't need to select a binsize
  - Easier to compare similarities and differences of different data distributions
  - Different classes of data distributions have distinct shapes
- The **cumulative distribution function** (CDF) lets us map between percentile rank and each value in a data column

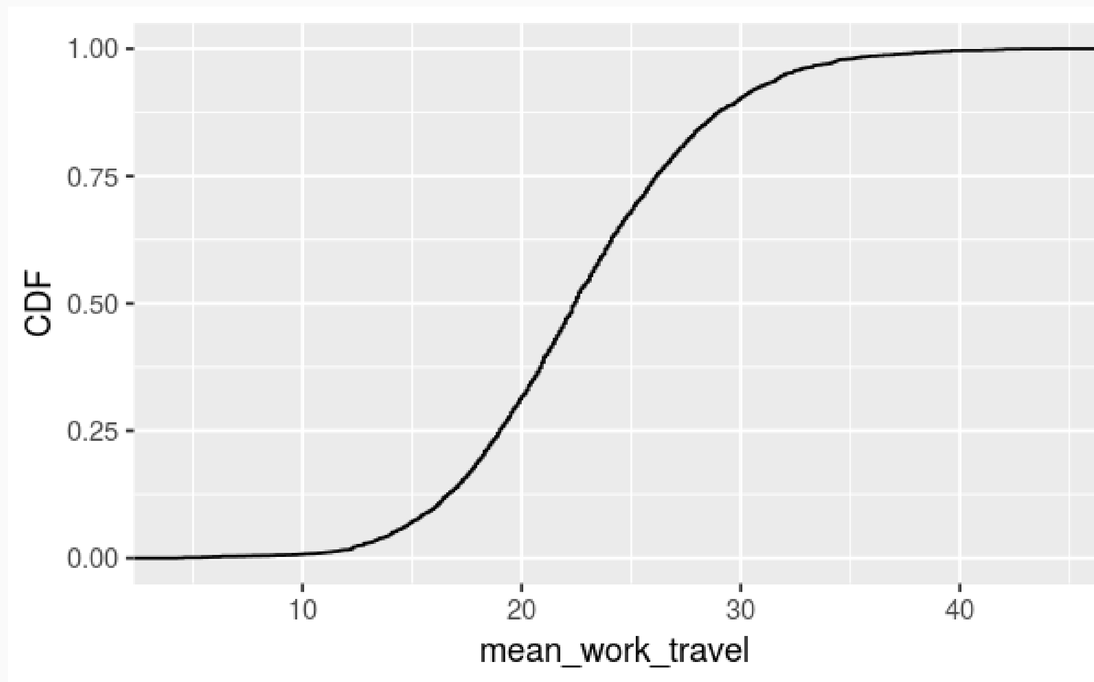
Creating CDFs in R

`ggplot2` comes with a handy convenience function `stat_ecdf()`, which lets you create CDF functions from your data

Creating CDFs in R

`ggplot2` comes with a handy convenience function `stat_ecdf()`, which lets you create CDF functions from your data

```
county %>%
  ggplot() +
  stat_ecdf(mapping = aes(x = mean_work_travel)) +
  labs(y = "CDF")
```



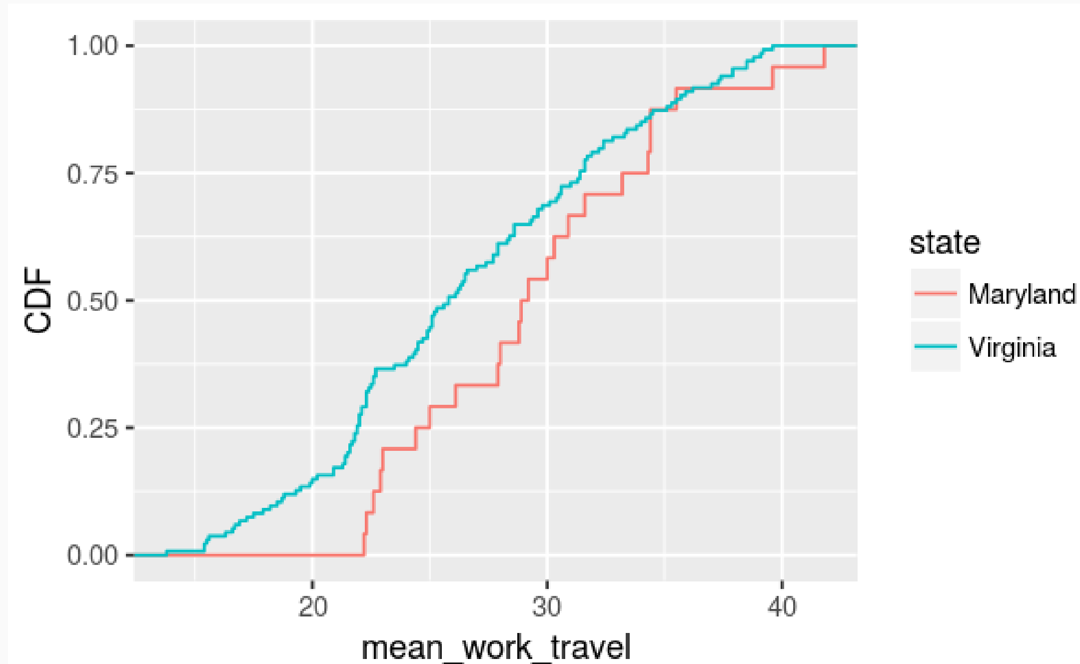
Creating CDFs in R

We can do all the usual operations, such as grouping by state

Creating CDFs in R

We can do all the usual operations, such as grouping by state

```
county %>%
  filter(state == "Virginia" | state == "Maryland") %>%
  ggplot() +
  stat_ecdf(mapping = aes(x = mean_work_travel, color = state)) +
  labs(y = "CDF")
```



Computing the CDF

To compute the CDF, we use the `cume_dist()` function along with `filter()`, `group_by()`, and `mutate()`:

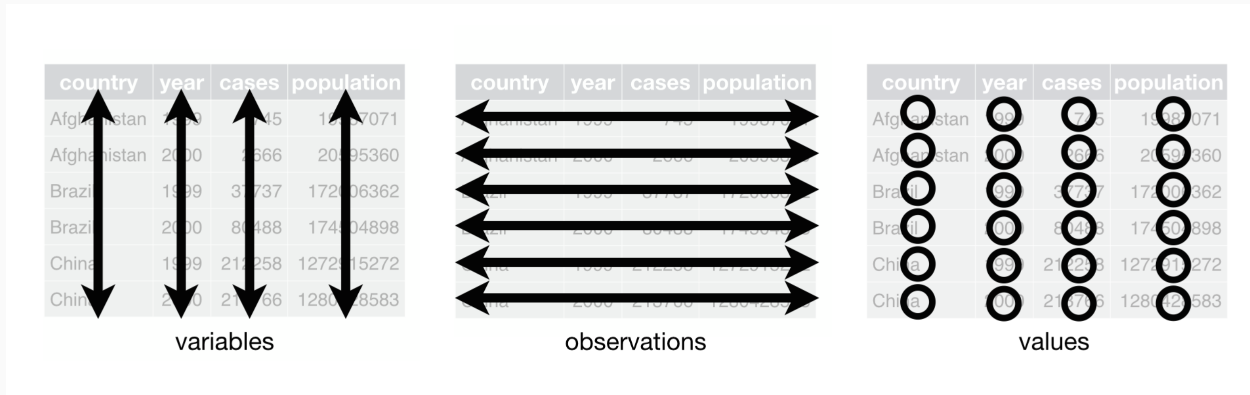
```
va_md_cdf_df <- county %>%
  filter(state == "Virginia" | state == "Maryland") %>%
  group_by(state) %>%
  mutate(cdf = cume_dist(mean_work_travel)) %>%
  select(state, mean_work_travel, cdf)
```

Get CDF data out of plot

state	mean_work_travel	cdf
Virginia	13.8	0.0074627
Virginia	15.4	0.0223881
Virginia	15.4	0.0223881
Virginia	15.5	0.0298507
Virginia	15.6	0.0373134
Virginia	16.3	0.0447761
Virginia	16.6	0.0522388
Virginia	16.7	0.0597015
Virginia	16.9	0.0671642
Virginia	17.2	0.0746269

Tidy data

Principles



1. Each variable must have its own column.
2. Each observation (case) must have its own row.
3. Each value must have its own cell.

Why should we care?

First, according to *R for Data Science*,

Why should we care?

First, according to *R for Data Science*,

1. There's a general advantage to picking one consistent way of storing data. If you have a consistent data structure, it's easier to learn the tools that work with it because they have an underlying uniformity.

2. There's a specific advantage to placing variables in columns because it allows R's vectorised nature to shine. As you learned in mutate and summary functions, most built-in R functions work with vectors of values. That makes transforming tidy data feel particularly natural.

Why should we care?

First, according to *R for Data Science*,

1. There's a general advantage to picking one consistent way of storing data. If you have a consistent data structure, it's easier to learn the tools that work with it because they have an underlying uniformity.

2. There's a specific advantage to placing variables in columns because it allows R's vectorised nature to shine. As you learned in mutate and summary functions, most built-in R functions work with vectors of values. That makes transforming tidy data feel particularly natural.

Translation: *Getting data into this form allows you to work on entire columns at a time using short and memorable commands*

Why should we care?

First, according to *R for Data Science*,

1. There's a general advantage to picking one consistent way of storing data. If you have a consistent data structure, it's easier to learn the tools that work with it because they have an underlying uniformity.

2. There's a specific advantage to placing variables in columns because it allows R's vectorised nature to shine. As you learned in mutate and summary functions, most built-in R functions work with vectors of values. That makes transforming tidy data feel particularly natural.

> ***Translation:*** *Getting data into this form allows you to work on entire columns at a time using short and memorable commands*

If you've programmed before, you are probably familiar with loops. In other languages, data manipulation may require you to tell your computer to scan the tabular dataset **one cell at a time**.

Why should we care?

First, according to *R for Data Science*,

1. There's a general advantage to picking one consistent way of storing data. If you have a consistent data structure, it's easier to learn the tools that work with it because they have an underlying uniformity.

2. There's a specific advantage to placing variables in columns because it allows R's vectorised nature to shine. As you learned in mutate and summary functions, most built-in R functions work with vectors of values. That makes transforming tidy data feel particularly natural.

> ***Translation:*** *Getting data into this form allows you to work on entire columns at a time using short and memorable commands*

If you've programmed before, you are probably familiar with loops. In other languages, data manipulation may require you to tell your computer to scan the tabular dataset **one cell at a time**. R can do this,

Why should we care?

First, according to *R for Data Science*,

1. There's a general advantage to picking one consistent way of storing data. If you have a consistent data structure, it's easier to learn the tools that work with it because they have an underlying uniformity.

2. There's a specific advantage to placing variables in columns because it allows R's vectorised nature to shine. As you learned in mutate and summary functions, most built-in R functions work with vectors of values. That makes transforming tidy data feel particularly natural.

> ***Translation:*** *Getting data into this form allows you to work on entire columns at a time using short and memorable commands*

If you've programmed before, you are probably familiar with loops. In other languages, data manipulation may require you to tell your computer to scan the tabular dataset **one cell at a time**. R can do this, but it's slow...

Why should we care?

First, according to *R for Data Science*,

1. There's a general advantage to picking one consistent way of storing data. If you have a consistent data structure, it's easier to learn the tools that work with it because they have an underlying uniformity.

2. There's a specific advantage to placing variables in columns because it allows R's vectorised nature to shine. As you learned in mutate and summary functions, most built-in R functions work with vectors of values. That makes transforming tidy data feel particularly natural.

> ***Translation:*** *Getting data into this form allows you to work on entire columns at a time using short and memorable commands*

If you've programmed before, you are probably familiar with loops. In other languages, data manipulation may require you to tell your computer to scan the tabular dataset **one cell at a time**. R can do this, but it's slow...

The "vectorized" tools of `tidyverse` are both faster and easier to understand!

Why should we care?

- There's a theoretical foundation to this, actually

Why should we care?

- There's a theoretical foundation to this, actually
- Closely related to the formalism of *relational databases*

Why should we care?

- There's a theoretical foundation to this, actually
- Closely related to the formalism of *relational databases*
- If you follow these rules, your data will be in Codd's 3rd normal form

Why should we care?

- There's a theoretical foundation to this, actually
- Closely related to the formalism of *relational databases*
- If you follow these rules, your data will be in Codd's 3rd normal form (if this means anything to you)

Why should we care?

- There's a theoretical foundation to this, actually
- Closely related to the formalism of *relational databases*
- If you follow these rules, your data will be in Codd's 3rd normal form (if this means anything to you)
- Helpful if you are working with a large or complex enough dataset that you need to store in a formal database, such as SQL databases (Postgresql, Mysql)

Why should we care?

- There's a theoretical foundation to this, actually
- Closely related to the formalism of *relational databases*
- If you follow these rules, your data will be in Codd's 3rd normal form (if this means anything to you)
- Helpful if you are working with a large or complex enough dataset that you need to store in a formal database, such as SQL databases (Postgresql, Mysql)
- Practically speaking, the tidying process makes the categories in your data more clear

Why should we care?

- There's a theoretical foundation to this, actually
- Closely related to the formalism of *relational databases*
- If you follow these rules, your data will be in Codd's 3rd normal form (if this means anything to you)
- Helpful if you are working with a large or complex enough dataset that you need to store in a formal database, such as SQL databases (Postgresql, Mysql)
- Practically speaking, the tidying process makes the categories in your data more clear
- It makes analysis much easier too, because you can easily subdivide your data by category, and apply transformations where needed

Why should we care?

- There's a theoretical foundation to this, actually
- Closely related to the formalism of *relational databases*
- If you follow these rules, your data will be in Codd's 3rd normal form (if this means anything to you)
- Helpful if you are working with a large or complex enough dataset that you need to store in a formal database, such as SQL databases (Postgresql, Mysql)
- Practically speaking, the tidying process makes the categories in your data more clear
- It makes analysis much easier too, because you can easily subdivide your data by category, and apply transformations where needed
- Provides a standardized, "best practices" way to structure and store our datasets

Why should we care?

- There's a theoretical foundation to this, actually
- Closely related to the formalism of *relational databases*
- If you follow these rules, your data will be in Codd's 3rd normal form (if this means anything to you)
- Helpful if you are working with a large or complex enough dataset that you need to store in a formal database, such as SQL databases (Postgresql, Mysql)
- Practically speaking, the tidying process makes the categories in your data more clear
- It makes analysis much easier too, because you can easily subdivide your data by category, and apply transformations where needed
- Provides a standardized, "best practices" way to structure and store our datasets
    - Note that you may not collect or input your data straight into tidy format

Tidying $\neq$ Cleaning

- Data tidying does **not** encompass the entire data cleaning process
- Data tidying only refers to reshaping things, such as moving columns and rows around
- Cleaning operations, such as correcting spelling errors, renaming variables, etc., is a separate topic

`tidyr()` package

Summary of `tidyr()` package

Summary of `tidyr()` package

- Functions (commands) that allow you to reshape data

Summary of `tidyr()` package

- Functions (commands) that allow you to reshape data
- Oriented towards the kinds of datasets we've worked with previously, each column may be a different data type (numeric, string, logical, etc)

Summary of `tidyr()` package

- Functions (commands) that allow you to reshape data
- Oriented towards the kinds of datasets we've worked with previously, each column may be a different data type (numeric, string, logical, etc)
- Functions (commands) are typed in a way that's very similar to the `dplyr` *verbs*, such as `filter()` and `mutate()`

Summary of `tidyr()` package

- Functions (commands) that allow you to reshape data
- Oriented towards the kinds of datasets we've worked with previously, each column may be a different data type (numeric, string, logical, etc)
- Functions (commands) are typed in a way that's very similar to the `dplyr` *verbs*, such as `filter()` and `mutate()`
- `tidyr` verbs

Summary of `tidyr()` package

- Functions (commands) that allow you to reshape data
- Oriented towards the kinds of datasets we've worked with previously, each column may be a different data type (numeric, string, logical, etc)
- Functions (commands) are typed in a way that's very similar to the `dplyr` *verbs*, such as `filter()` and `mutate()`
- `tidyr` verbs
    - `gather()`: transforms wide data to narrow data

Summary of `tidyr()` package

- Functions (commands) that allow you to reshape data
- Oriented towards the kinds of datasets we've worked with previously, each column may be a different data type (numeric, string, logical, etc)
- Functions (commands) are typed in a way that's very similar to the `dplyr` *verbs*, such as `filter()` and `mutate()`
- `tidyr` verbs
  - `gather()`: transforms wide data to narrow data
  - `spread()`: transforms narrow data to wide data

Summary of `tidyr()` package

- Functions (commands) that allow you to reshape data
- Oriented towards the kinds of datasets we've worked with previously, each column may be a different data type (numeric, string, logical, etc)
- Functions (commands) are typed in a way that's very similar to the `dplyr` *verbs*, such as `filter()` and `mutate()`
- `tidyr` verbs
    - `gather()`: transforms wide data to narrow data
    - `spread()`: transforms narrow data to wide data
    - `separate()`: make multiple columns out of a single column

Summary of `tidyr()` package

- Functions (commands) that allow you to reshape data
- Oriented towards the kinds of datasets we've worked with previously, each column may be a different data type (numeric, string, logical, etc)
- Functions (commands) are typed in a way that's very similar to the `dplyr` *verbs*, such as `filter()` and `mutate()`
- `tidyr` verbs
  - `gather()`: transforms wide data to narrow data
  - `spread()`: transforms narrow data to wide data
  - `separate()`: make multiple columns out of a single column
  - `unite()`: make a single column out of multiple columns

Simple examples from textbook

Follow along in RStudio

Credits

- Slides in the section Statistical distributions adapted from the Chapter 1 OpenIntro Statistics slides developed by Mine Çetinkaya-Rundel and made available under the CC BY-SA 3.0 license.