Section 001 CDS 101 *Introduction to Computational and Data Sciences*

Class 21: Statistical distributions II

April 10, 2018

General

Annoucements

- Reading for next class: *Introductory Statistics with Randomization and Simulation*
    - From chapter 2: from the beginning through to the end of section 2.2
- Homework 3 posted, due next Monday, April 16th by 11:59pm.

Statistical computations in R

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
- `quantile()`: Computes quantiles (percentiles)

Using the statistical functions

Using the statistical functions

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

Using the statistical functions

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

Using the statistical functions

- Every function except `quantile()` 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 `quantile()` 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

- `quantile()` can output one value at a time, or many values

Using the statistical functions

- `quantile()` can output one value at a time, or many values
- To be able to use it in `summarize()`, stick with one value at a time

Using the statistical functions

- `quantile()` can output one value at a time, or many values
- To be able to use it in `summarize()`, stick with one value at a time

```
county %>%
  summarize(
    Q1 = quantile(mean_work_travel, probs = c(0.25), type = 1),
    Q2 = quantile(mean_work_travel, probs = c(0.50), type = 1),
    Q3 = quantile(mean_work_travel, probs = c(0.75), type = 1),
    Q4 = quantile(mean_work_travel, probs = c(1.00), type = 1))
```

Using the statistical functions

- `quantile()` can output one value at a time, or many values
- To be able to use it in `summarize()`, stick with one value at a time

```
county %>%
  summarize(
    Q1 = quantile(mean_work_travel, probs = c(0.25), type = 1),
    Q2 = quantile(mean_work_travel, probs = c(0.50), type = 1),
    Q3 = quantile(mean_work_travel, probs = c(0.75), type = 1),
    Q4 = quantile(mean_work_travel, probs = c(1.00), type = 1))
```

Q1	Q2	Q3	Q4
19	22.4	26.1	44.2

From histograms to probability mass functions

Data distributions

- Early on in the course, we learned that histograms via `geom_histogram()` are a convenient way to represent numerical data in a single column (variable)

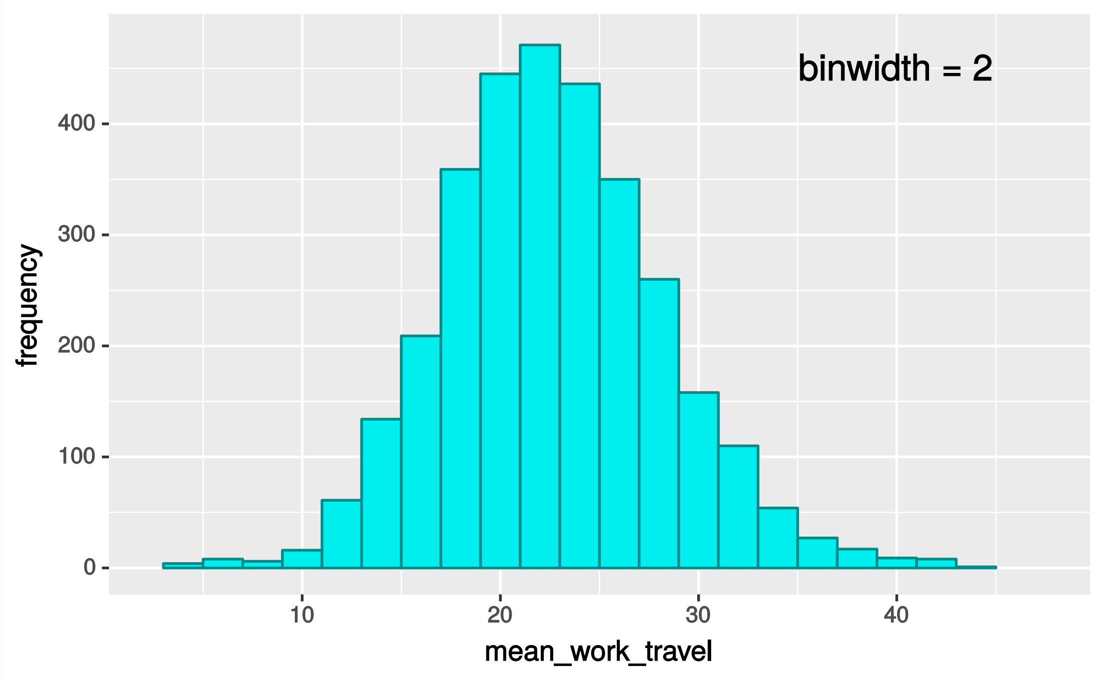
Data distributions

- Early on in the course, we learned that histograms via `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
...

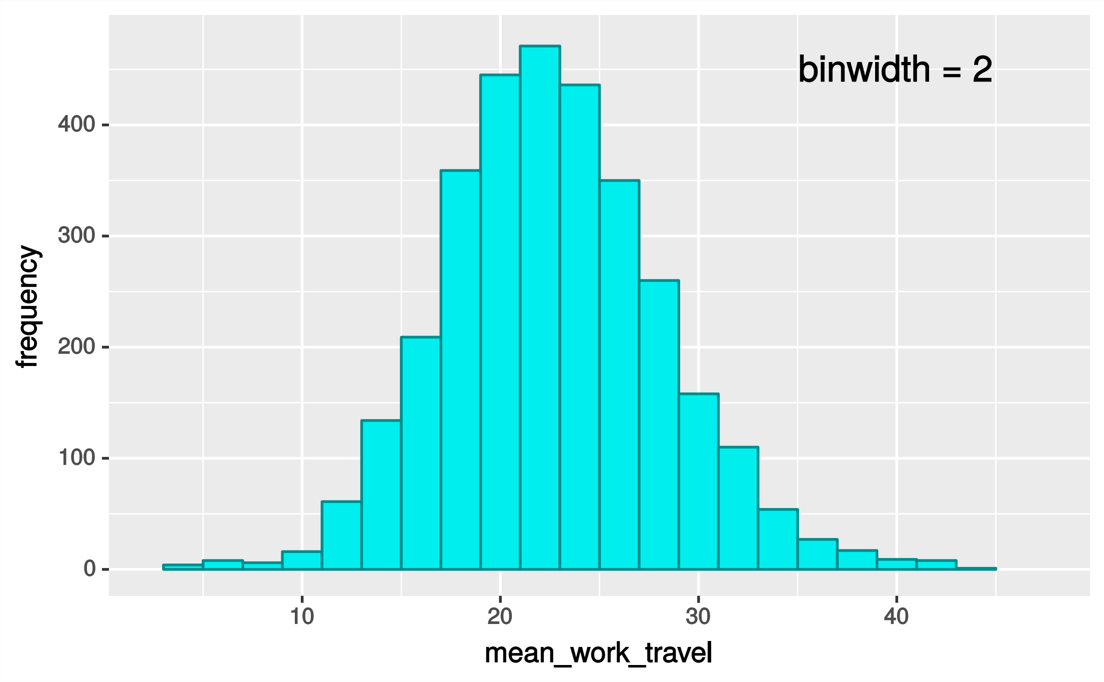
Data distributions

- Early on in the course, we learned that histograms via `geom_histogram()` are a convenient way to represent numerical data in a single column (variable)



Data distributions

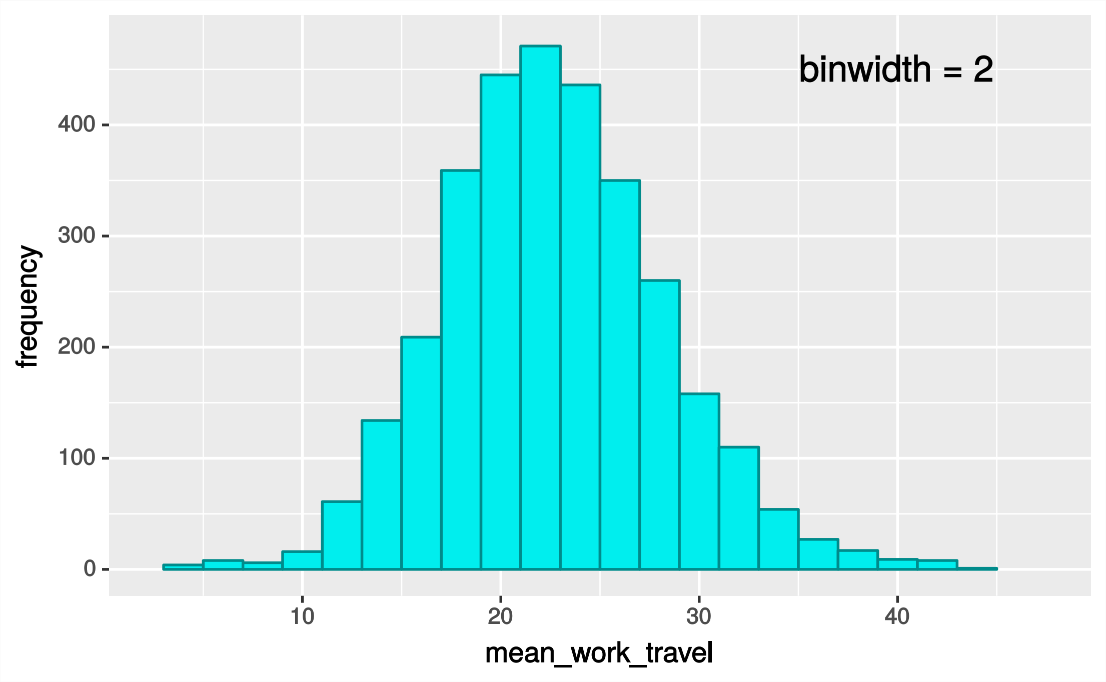
- Early on in the course, we learned that histograms via `geom_histogram()` are a convenient way to represent numerical data in a single column (variable)



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

Data distributions

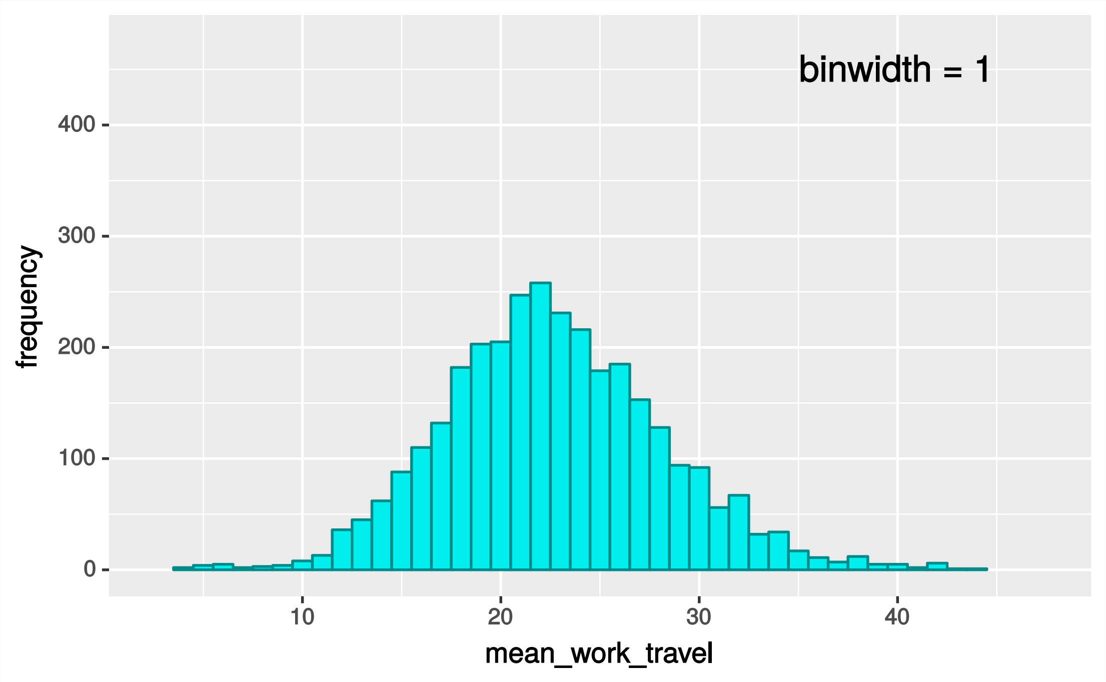
- Early on in the course, we learned that histograms via `geom_histogram()` are a convenient way to represent numerical data in a single column (variable)



- 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

- Early on in the course, we learned that histograms via `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 skipped over the question of how to compare distributions with varying numbers of observations

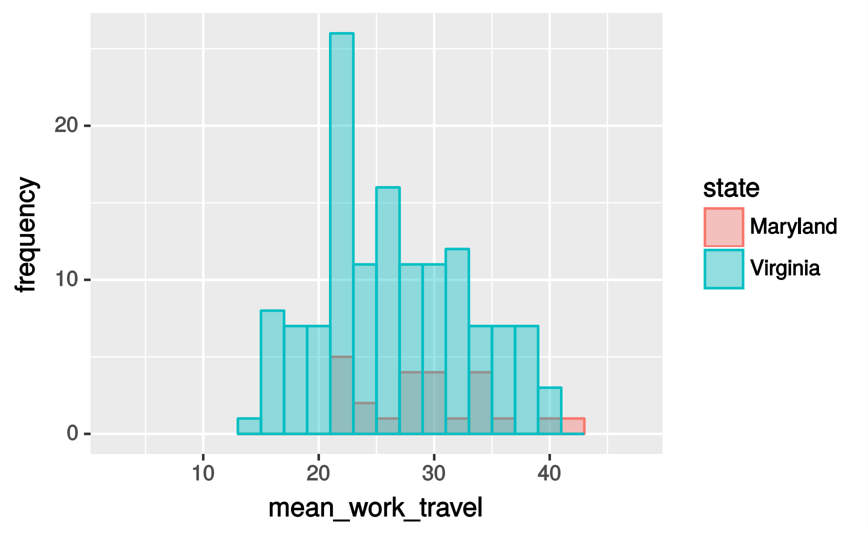
Comparing distributions with unequal observations

- So far, we've 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

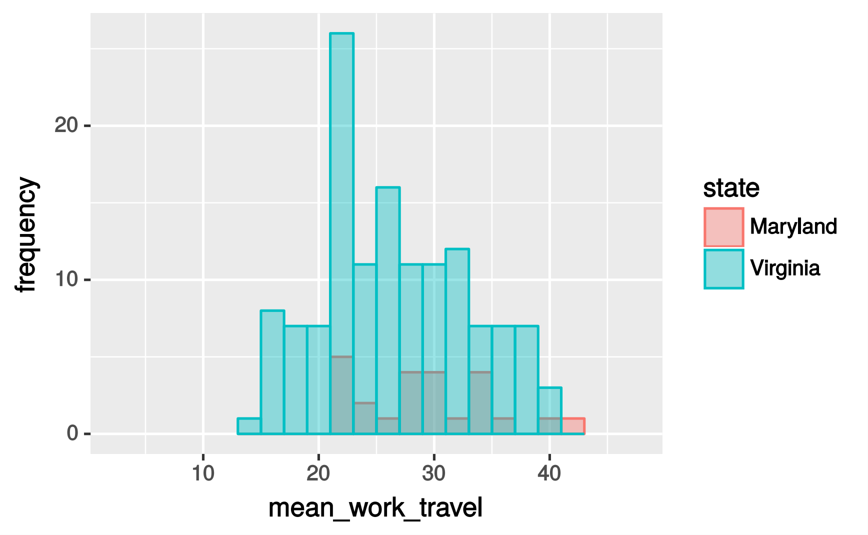
- So far, we've skipped over the question of how to compare distributions with varying numbers of observations

- 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

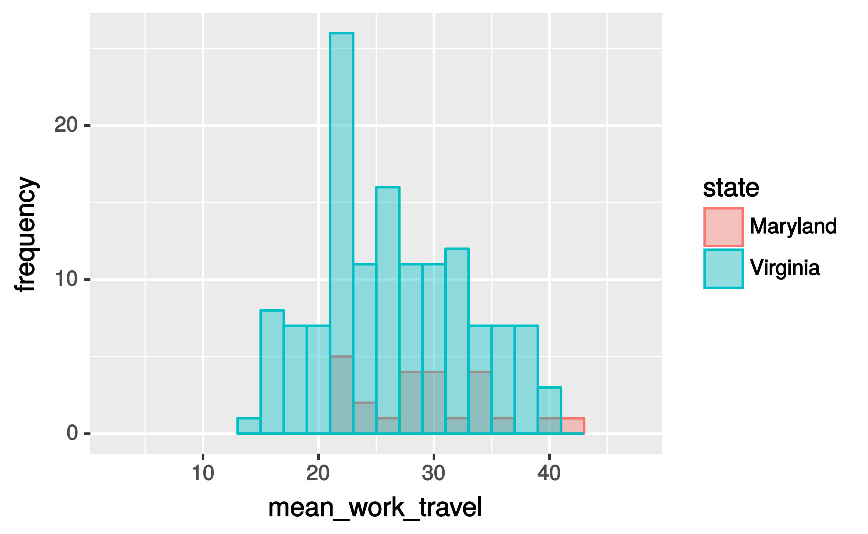
- So far, we've skipped over the question of how to compare distributions with varying numbers of observations
- In our current example of average times to travel to work, we can group the data by state and compare Virginia to Maryland



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

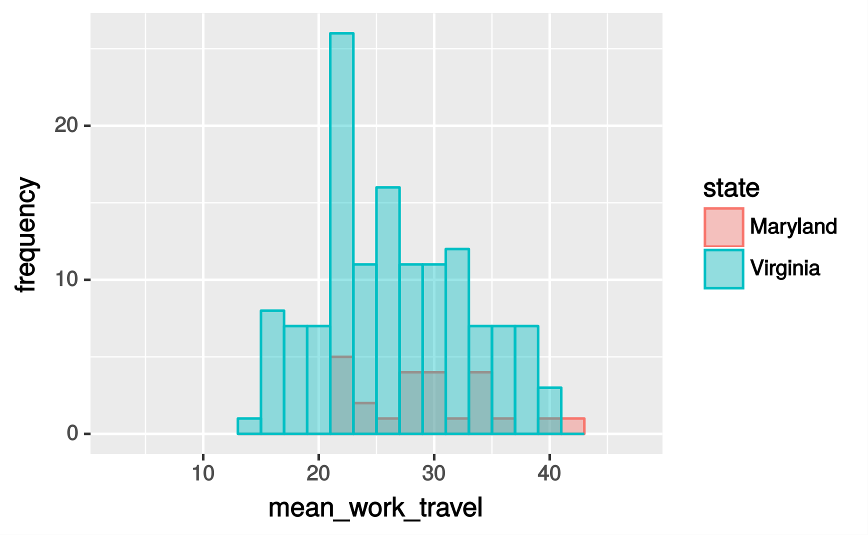
Comparing distributions with unequal observations

- So far, we've 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 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

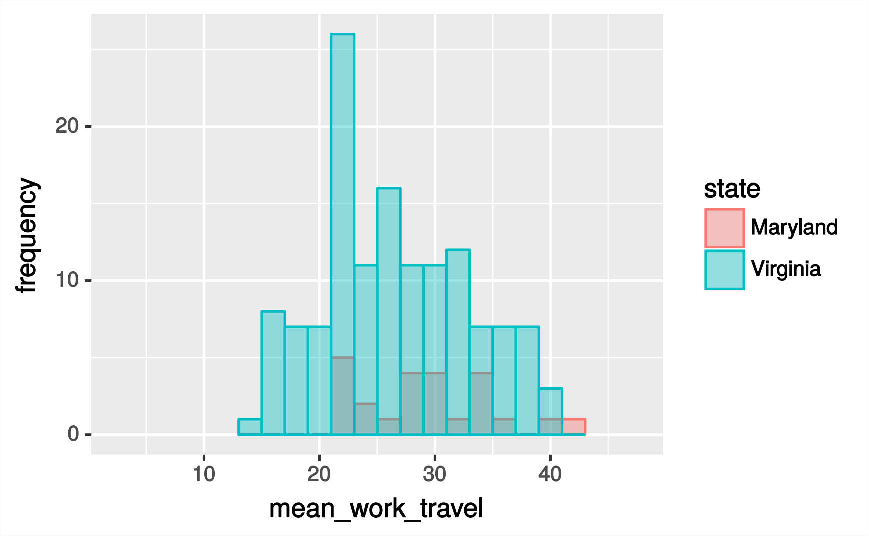


- In the dataset, Virginia has 134 counties compared to Maryland's 24 counties

Comparing distributions with unequal observations

- So far, we've 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

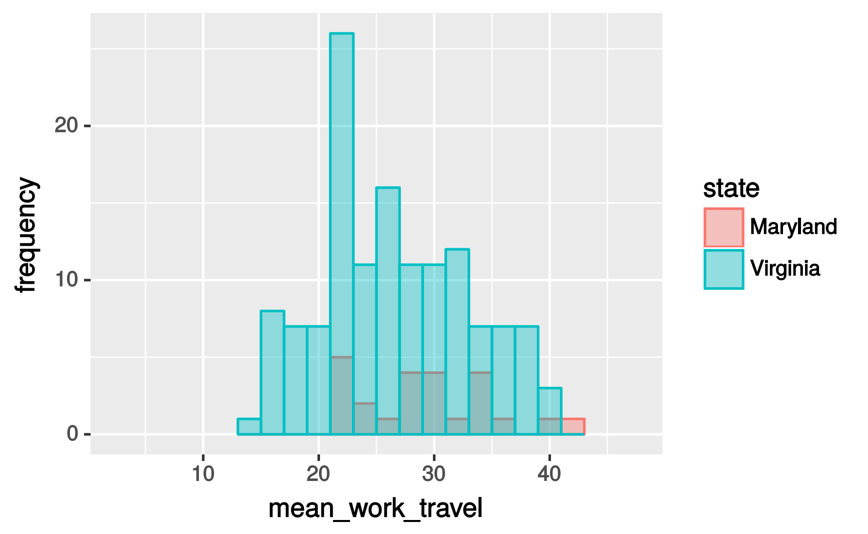
- Normalization is straightforward, just divide the frequency count in each "bucket" by the total number of observations in the histogram

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

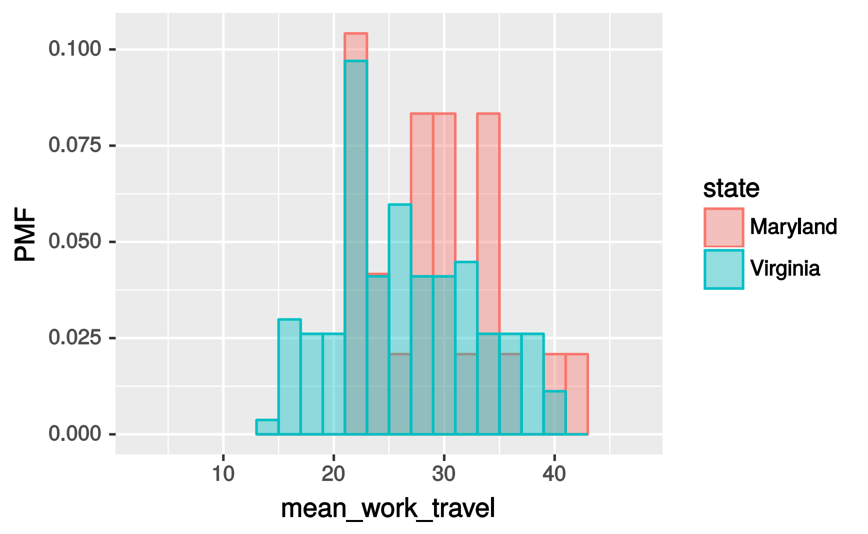
From frequency to probability

- 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

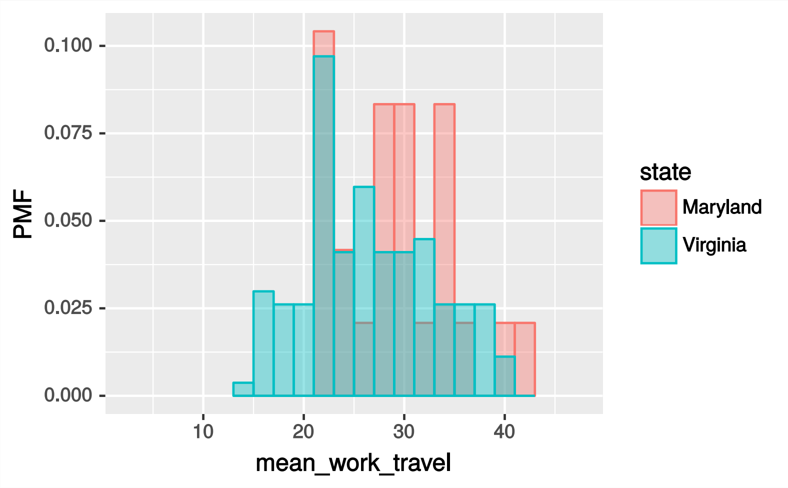


From frequency to probability

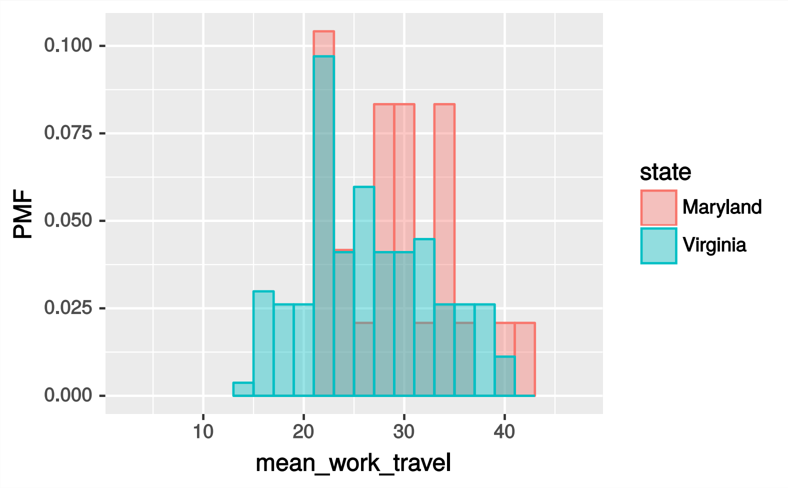
- Normalization is straightforward, just divide the frequency count in each "bucket" by the total number of observations in the histogram
- If you group by categories, that you should divide by the number of observations in each group
- To normalize the histograms from the prior example, we need to divide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



Probability mass function (PMF)

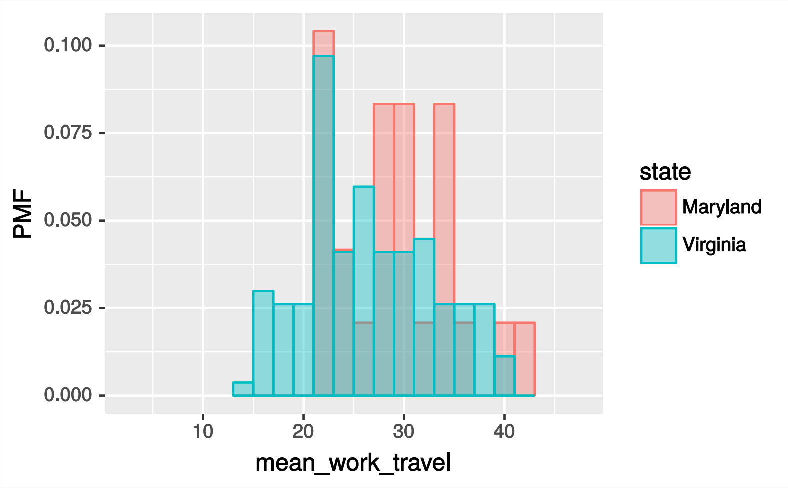


Probability mass function (PMF)



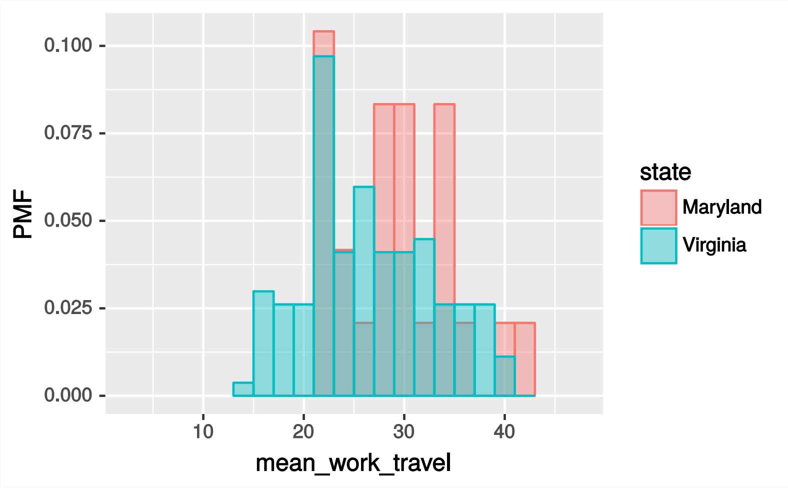
- Just like a histogram, except that the bar heights reflect **probabilities** instead of **frequency counts**

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

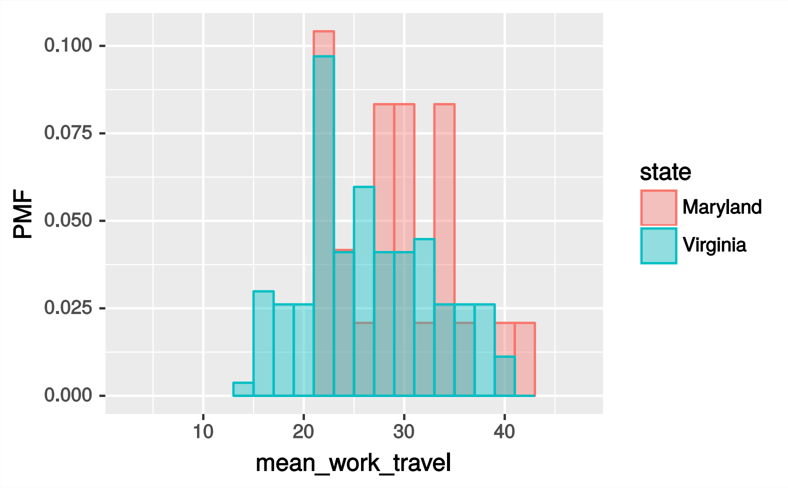
Probability mass function (PMF)



- 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),
    binwidth = 2, center = 0, position = "identity", alpha = 0.4) +
  labs(y = "frequency") +
  coord_cartesian(xlim = c(2.5, 47.5))
```

Creating PMFs in R

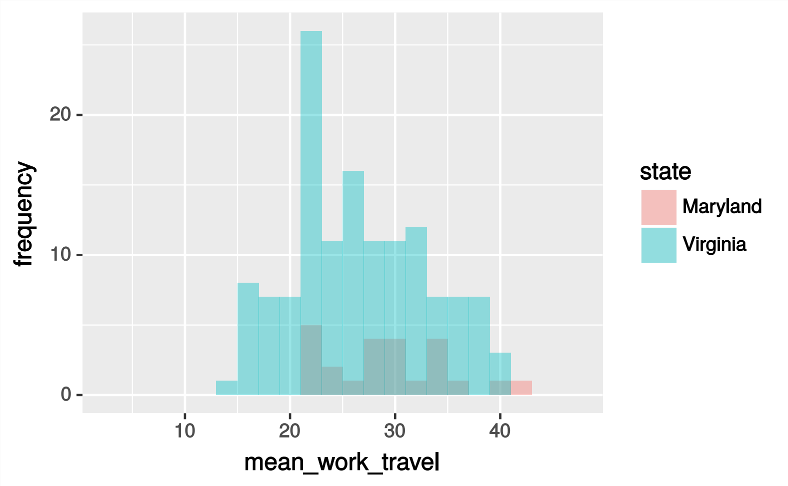
- With `ggplot2`, it's straightforward to convert a histogram into a PMF

```
county %>%
  filter(state == "Virginia" | state == "Maryland") %>%
  ggplot() +
  geom_histogram(
    mapping = aes(x = mean_work_travel, fill = state),
    binwidth = 2, center = 0, position = "identity", alpha = 0.4) +
  labs(y = "frequency") +
  coord_cartesian(xlim = c(2.5, 47.5))
```

Creating PMFs in R

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

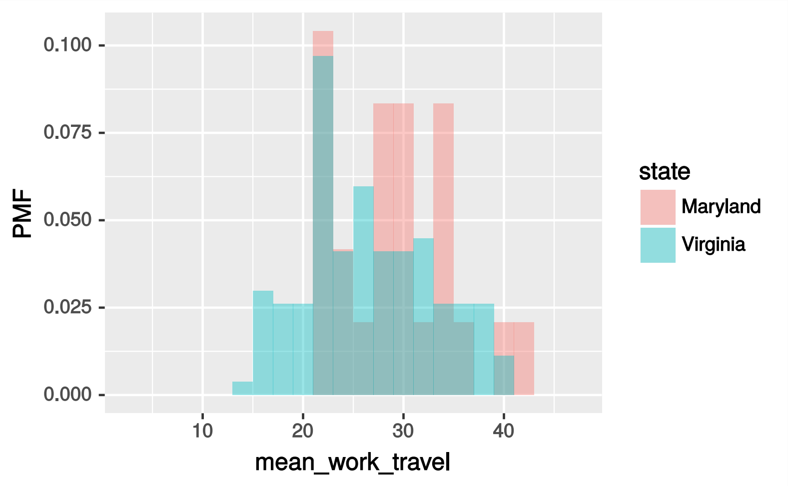
```
county %>%
  filter(state == "Virginia" | state == "Maryland") %>%
  ggplot() +
  geom_histogram(
    mapping = aes(x = mean_work_travel, fill = state),
    binwidth = 2, center = 0, position = "identity", alpha = 0.4) +
  labs(y = "frequency") +
  coord_cartesian(xlim = c(2.5, 47.5))
```



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),
    binwidth = 2, center = 0, position = "identity", alpha = 0.4) +
  labs(y = "PMF") +
  coord_cartesian(xlim = c(2.5, 47.5))
```



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_tibble()
```

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

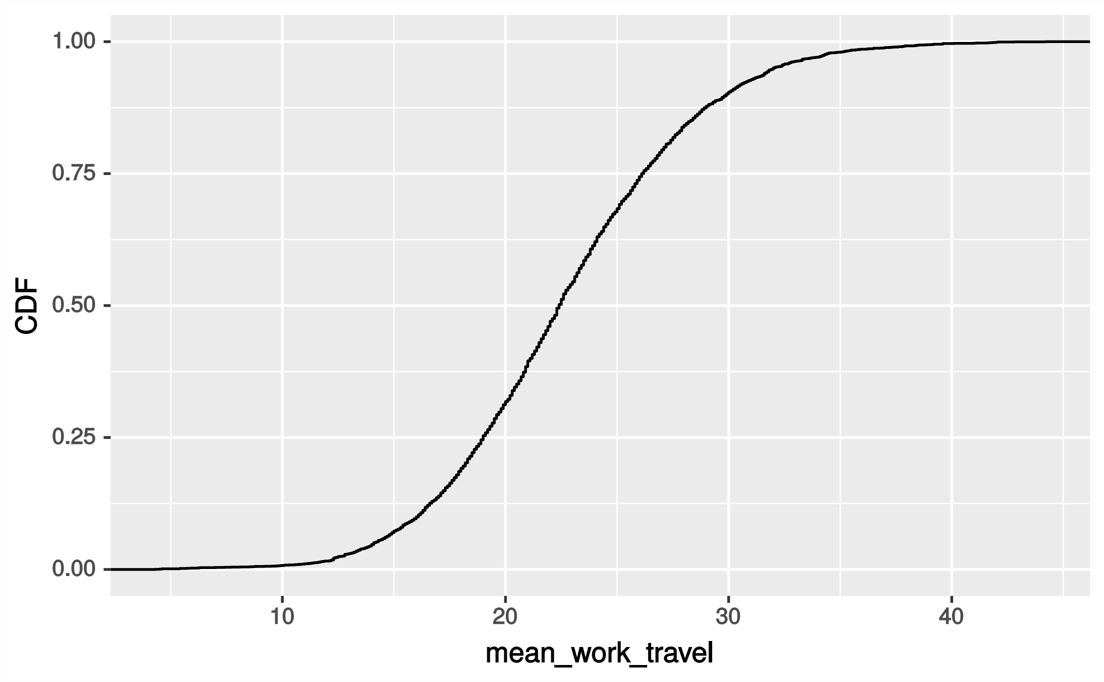
Creating CDFs in R

`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")
```



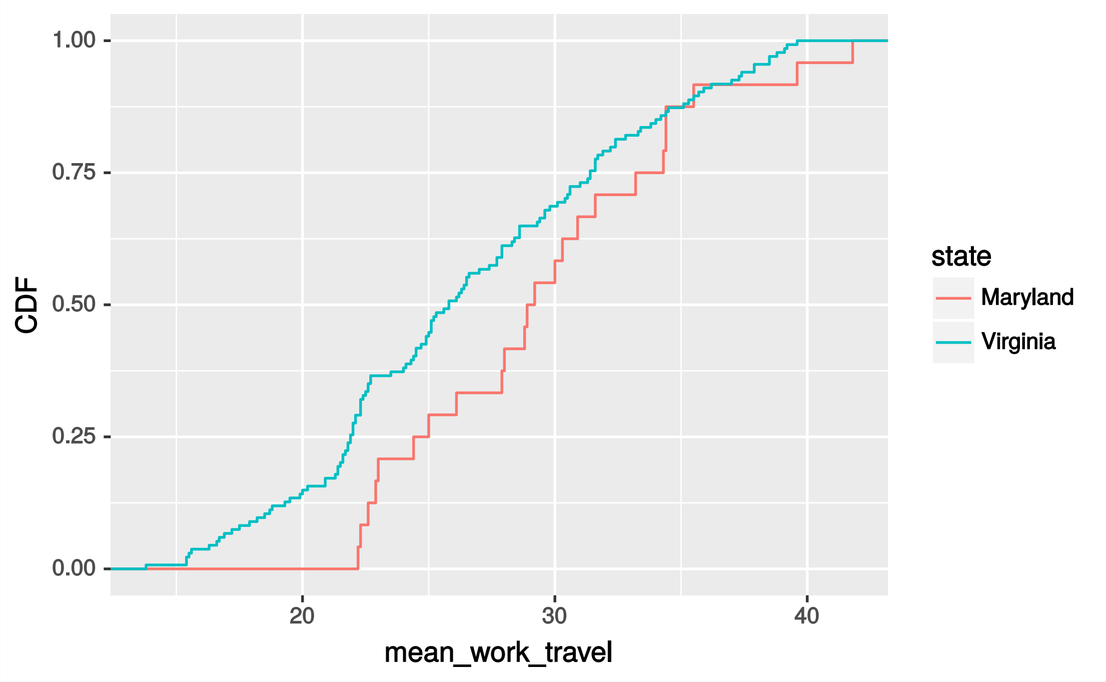
Creating CDFs in R

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")
```



Get CDF data out of plot

Assign the plot to a variable:

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

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

```
va_md_cdf_df <- va_md_cdf_figure %>%
  ggplot_build() %>%
  pluck("data", 1) %>%
  as_tibble() %>%
  select(group, x, y) %>%
  rename(mean_work_travel = "x", cdf = "y", state = "group") %>%
  mutate(state = recode(state, `1` = "Maryland", `2` = "Virginia")) %>%
  arrange(desc(state), mean_work_travel)
```

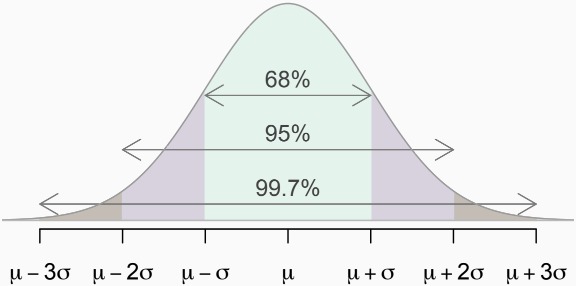
Get CDF data out of plot

state	mean_work_travel	cdf
Virginia	-Inf	0
Virginia	13.8	0.00746268656716418
Virginia	15.4	0.0223880597014925
Virginia	15.5	0.0298507462686567
Virginia	15.6	0.0373134328358209
Virginia	16.3	0.0447761194029851
Virginia	16.6	0.0522388059701493
Virginia	16.7	0.0597014925373134
Virginia	16.9	0.0671641791044776
Virginia	17.2	0.0746268656716418
…	…	…

Analyzing the normal distribution

68-95-99.7 Rule

- For nearly normally distributed data,
  - about 68% falls within 1 SD of the mean,
  - about 95% falls within 2 SD of the mean,
  - about 99.7% falls within 3 SD of the mean.
- It is possible for observations to fall 4, 5, or more standard deviations away from the mean, but these occurrences are very rare if the data are nearly normal.



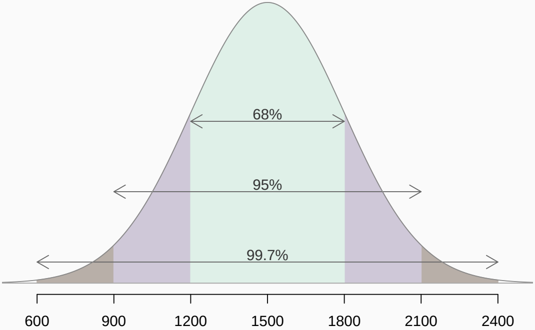
Describing variability using the 68-95-99.7 Rule

SAT scores are distributed nearly normally with mean 1500 and standard deviation 300.

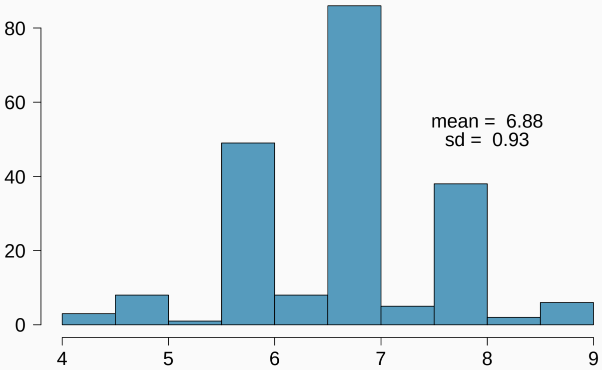
Describing variability using the 68-95-99.7 Rule

SAT scores are distributed nearly normally with mean 1500 and standard deviation 300.

- ~68% of students score between 1200 and 1800 on the SAT.
- ~95% of students score between 900 and 2100 on the SAT.
- ~99.7% of students score between 600 and 2400 on the SAT.



Number of hours of sleep on school nights



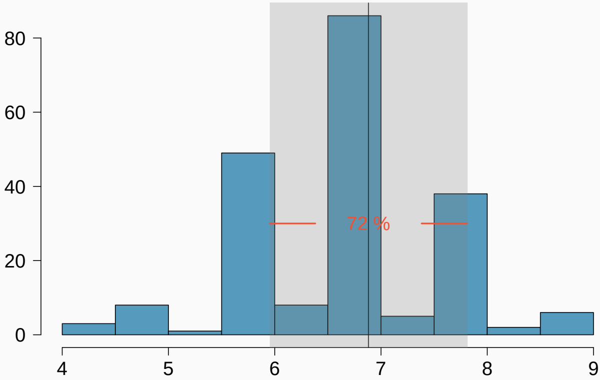
- Mean = 6.88 hours, SD = 0.92 hrs

  72% of the data are within 1 SD of the mean: 6.88 ± 0.93

  92% of the data are within 1 SD of the mean: $6.88 \pm 2 \times 0.93$

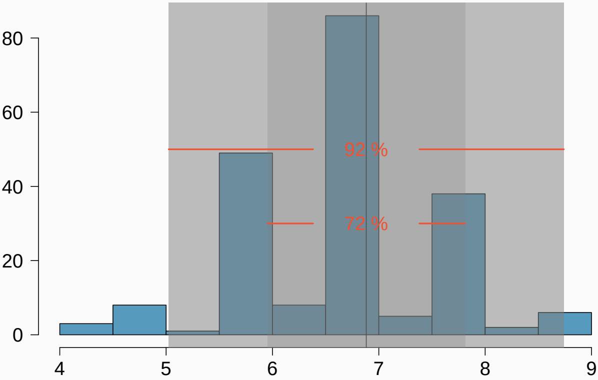
  99% of the data are within 1 SD of the mean: $6.88 \pm 3 \times 0.93$

Number of hours of sleep on school nights



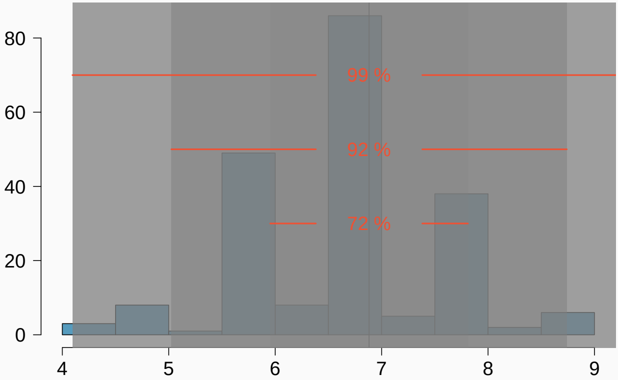
- Mean = 6.88 hours, SD = 0.92 hrs
- 72% of the data are within 1 SD of the mean: 6.88 ± 0.93
- 92% of the data are within 1 SD of the mean: $6.88 \pm 2 \times 0.93$
- 99% of the data are within 1 SD of the mean: $6.88 \pm 3 \times 0.93$

Number of hours of sleep on school nights



- Mean = 6.88 hours, SD = 0.92 hrs
- 72% of the data are within 1 SD of the mean: 6.88 ± 0.93
- 92% of the data are within 1 SD of the mean: $6.88 \pm 2 \times 0.93$

Number of hours of sleep on school nights

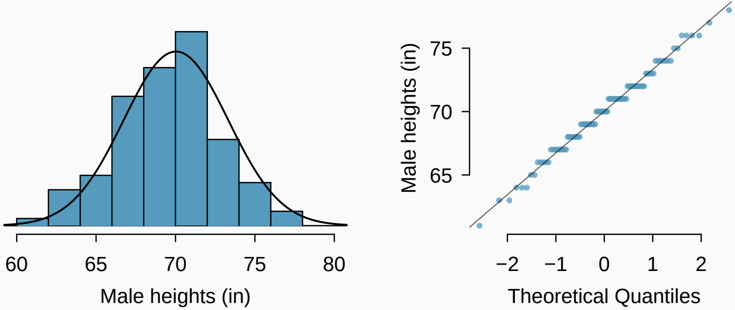


- Mean = 6.88 hours, SD = 0.92 hrs
- 72% of the data are within 1 SD of the mean: 6.88 ± 0.93
- 92% of the data are within 1 SD of the mean: $6.88 \pm 2 \times 0.93$
- 99% of the data are within 1 SD of the mean: $6.88 \pm 3 \times 0.93$

Evaluating the normal approximation

Normal probability plot

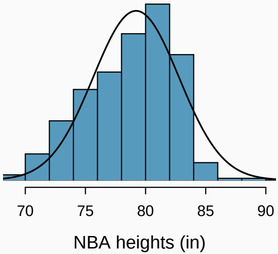
A histogram and *normal probability plot* of a sample of 100 male heights.

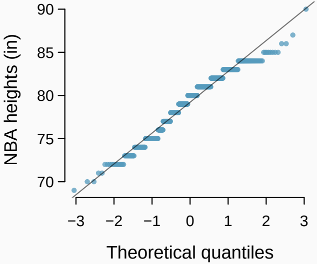


Anatomy of a normal probability plot

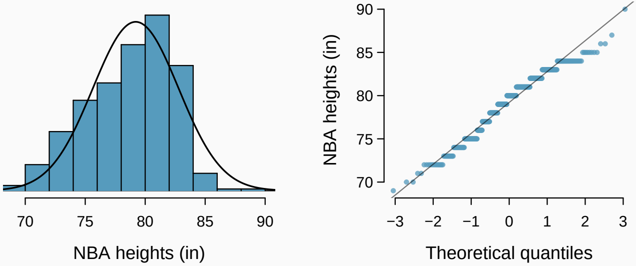
- Data are plotted on the y-axis of a normal probability plot, and theoretical quantiles (following a normal distribution) on the x-axis.
- If there is a linear relationship in the plot, then the data follow a nearly normal distribution.
- Constructing a normal probability plot requires calculating percentiles and corresponding z-scores for each observation, which is tedious. Therefore we generally rely on software when making these plots.

Below is a histogram and normal probability plot for the NBA heights from the 2008-2009 season. Do these data appear to follow a normal distribution?



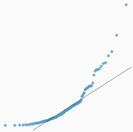


Below is a histogram and normal probability plot for the NBA heights from the 2008-2009 season. Do these data appear to follow a normal distribution?

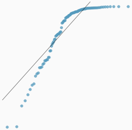


Why do the points on the normal probability have jumps?

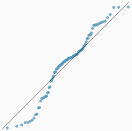
Normal probability plot and skewness



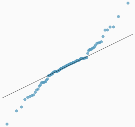
Right skew - Points bend up and to the left of the line.



Left skew- Points bend down and to the right of the line.



Short tails (narrower than the normal distribution) - Points follow an S shaped-curve.



Long tails (wider than the normal distribution) - Points start below the line, bend to follow it, and end above it.

Central limit theorem

Central limit theorem

Review the Central Limit Theorem animation on Seeing Theory

Credits

The slides with blue headers originate from the following source:

- The Chapter 3 OpenIntro Statistics slides developed by Mine Çetinkaya-Rundel and made available under the CC BY-SA 3.0 license.