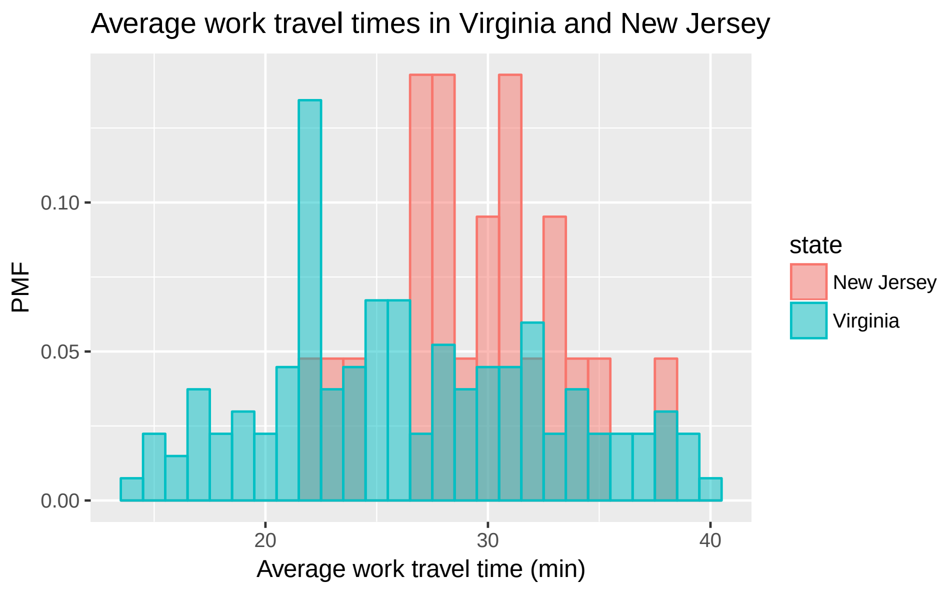
Cumulative distribution functions

```r
library(tidyverse)
county_complete <- read_rds(
  path = url("http://spring18.cds101.com/files/datasets/county_complete.rds"))
```

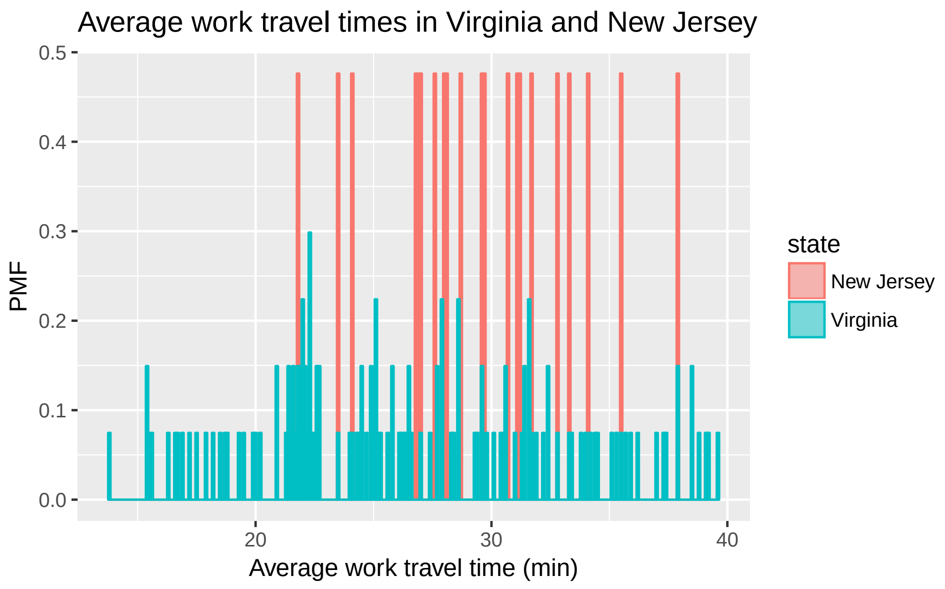
The limits of probability mass functions

Probability mass functions (PMFs) work well if the number of unique values is small. But as the number of unique values increases, the probability associated with each value gets smaller and the effect of random noise increases.

Let's recall that, in the previous reading, we plotted and compared PMFs of the average work travel time in Virginia and New Jersey, which resulted in this figure:



What happens if we choose `binwidth = 0.1` for plotting the `mean_work_travel` distribution? The values in `mean_work_travel` are reported to the first decimal place, so `binwidth = 0.1` does not "smooth out" the data. This increases the number of distinct values in `mean_work_travel` from 41 to 304. The comparison between the Virginia and New Jersey PMFs will then look like this:



This visualization has a lot of spikes of similar heights, which makes this difficult to interpret and limits its usefulness. Also, it can be hard to see overall patterns; for example, what is the approximate difference in means between these two distributions?

This illustrates the tradeoff when using histograms and PMFs for visualizing single variables. If we smooth things out by using larger bin sizes, then we can lose information that may be useful. On the other hand, using small bin sizes creates plots like the one above, which is of limited (if any) utility.

An alternative that avoids these problems is the cumulative distribution function (CDF), which we turn to describing next. But before we discuss CDFs, we first have to understand the concept of percentiles.

Percentiles

If you have taken a standardized test, you probably got your results in the form of a raw score and a **percentile rank**. In this context, the percentile rank is the fraction of people who scored lower than you (or the same). So if you are "in the 90th percentile", you did as well as or better than 90% of the people who took the exam.

As an example, say that you and 4 other people took a test and received the following scores:

```
55   66   77   88   99
```

If you received the score of 88, then what is your percentile rank? We can calculate it as follows

```
test_scores <- tribble(
  ~score,
  55,
  66,
  77,
  88,
  99)

number_of_tests <- test_scores %>%
  count() %>%
  pull(n)
```

```r
number_of_lower_scores <- test_scores %>%
  filter(score <= 88) %>%
  count() %>%
  pull(n)

percentile_rank <- 100.0 * number_of_lower_scores / number_of_tests
```

From this, we find that the percentile rank for a score of 88 is 80. Mathematically, the calculation is $100 \times \frac{4}{5} = 80$.

As you can see, if you are given a value, it is easy to find its percentile rank; going the other way is slightly harder. One way to do this is to sort the scores and find the row number that corresponds to a percentile rank. To find the row number, divide the total number of scores by 100, multiply that number by the desired percentile rank, and then *round up* to the nearest integer value. The rounding up operation can be handled via the `ceiling()` function. So, for our example, the value with percentile rank 55 is:

```r
percentile_rank_row_number <- ceiling(55 * number_of_tests / 100)
test_scores %>%
  arrange(score) %>%
  slice(percentile_rank_row_number)
```

score
77

The result of this calculation is called a **percentile**. So this means that, in the distribution of exam scores, the 55th percentile corresponds to a score of 77.

In R, there is a function called `quantile()` that can do the above calculation automatically, although you need to take care with the inputs. Let's first show what happens when we aren't careful. We might think that we can calculate the 55th percentile by running:

```r
test_scores %>%
  pull(score) %>%
  quantile(probs = c(0.55))
```

	x
55%	79.2

We get a score of 79.2, which isn't in our dataset. This happens because `quantile()` interpolates between the scores by default. Sometimes you will want this behavior, other times you will not. When the dataset is this small, it doesn't make as much sense to permit interpolation, as it can be based on rather aggressive assumptions about what intermediate scores might look like. To tell `quantile()` to compute scores in the same manner as we did above, add the input `type = 1`:

```r
test_scores %>%
  pull(score) %>%
  quantile(probs = c(0.55), type = 1)
```

	x
55%	77

This, as expected, agrees with the manual calculation.

It is worth emphasizing that the difference between "percentile" and "percentile rank" can be confusing, and people do not always use the terms precisely. To summarize, if we want to know the percentage of people obtained scores equal to or lower than ours, then we are computing a percentile rank. If we start with a percentile, then we are computing the score in the distribution that corresponds with it.

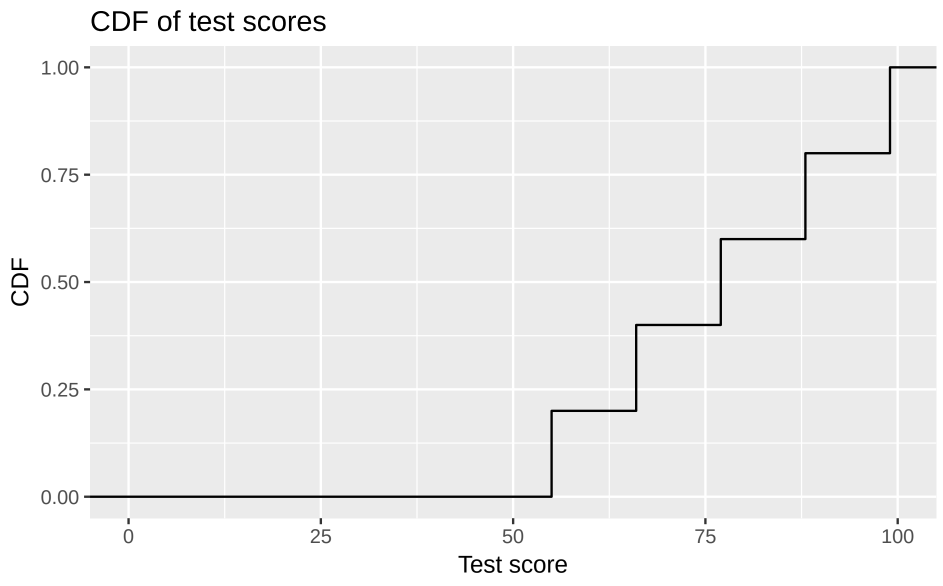
CDFs

Now that we understand percentiles and percentile ranks, we are ready to tackle the **cumulative distribution function** (CDF). The CDF is the function that maps from a value to its percentile rank. To find the CDF for any particular value in our distribution, we compute the fraction of values in the distribution less than or equal to our selected value. Computing this is similar to how we calculated the percentile rank, except that the result is a probability in the range 0–1 rather than a percentile rank in the range 0–100. For our test scores example, we can manually compute the CDF in the following way:

```
test_scores_cdf <- test_scores %>%
  arrange(score) %>%
  mutate(cdf = row_number() / n())
```

score	cdf
55	0.2
66	0.4
77	0.6
88	0.8
99	1.0

The visualization of the CDF looks like:



As you can see, the CDF of a sample looks like a sequence of steps. Appropriately enough, this is called a step function, and the CDF of *any* sample is a step function.

Also note that we can evaluate the CDF for any value, not just values that appear in the sample. If we select a value that is less than the smallest value in the sample, then the CDF is 0. If we select a value that is greater than the largest value, then the CDF is 1.

Representing CDFs

While it's good to know how to manually compute the CDF, R provides the `ecdf()` function, which constructs the CDF of a sample automatically. Let's return to the average work travel times dataset we used when discussing the PMF and compute the CDF of the full distribution (no grouping by states). We do this as follows:

```
mean_work_travel_ecdf <- county_complete %>%
  pull(mean_work_travel) %>%
  ecdf()
```

Now we can input an arbitrary travel time and find the percentile. For example, the percentile for an average work travel time of 30 minutes is:

```
mean_work_travel_ecdf(30)
```

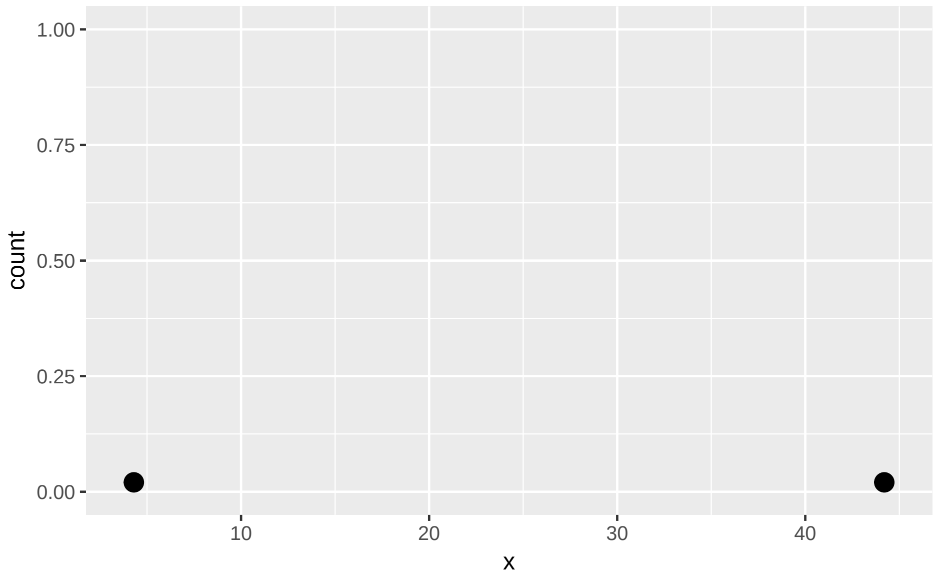
```
## [1] 0.9045498
```

Thus, 30 minutes corresponds to the 90th percentile.

We can also use this to create a plot. To do this, we need to generate a sequence of travel times and calculate the CDF for each. A recommended way to do this is to find the minimum and maximum values of the distribution using `min()` and `max()`, and then use `seq()` to generate a long list of values that sit inbetween the minimum and maximum. Let's show a couple examples so it's clear what we're doing. First we find the minimum and maximum:

```
mean_work_travel_min <- county_complete %>%
  pull(mean_work_travel) %>%
  min()

mean_work_travel_max <- county_complete %>%
  pull(mean_work_travel) %>%
  max()
```

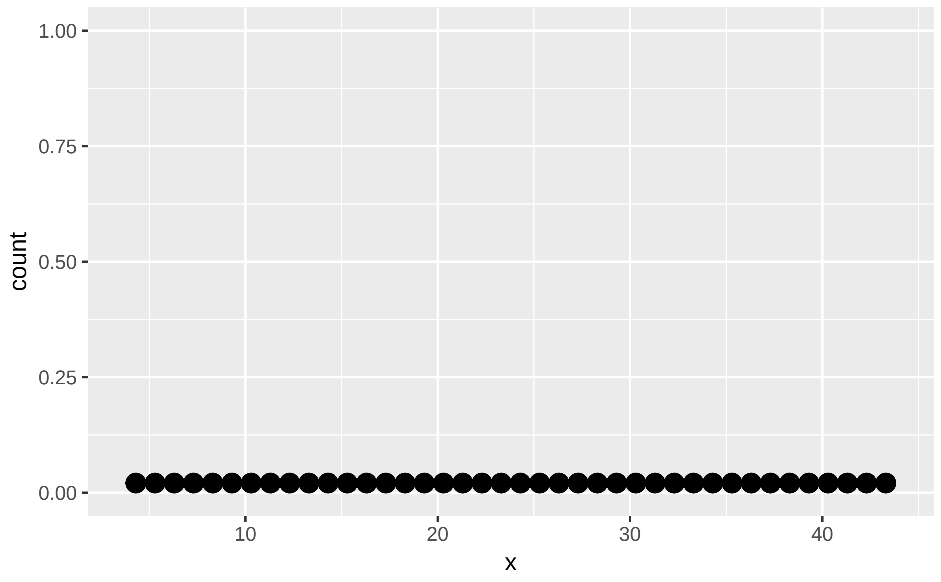
By themselves, these correspond to two points along the horizontal axis, like so



To generate horizontal values that increase by 1 between the minimum and maximum, we use `seq()` as follows:

```r
mean_work_travel_range1 <- seq(
  from = mean_work_travel_min, to = mean_work_travel_max, by = 1)
```

Visually, this would look like



Generally, it's better to use a smaller incremental value inside `seq()` in order to make smoother figures. Let's use `by = 0.1`:

```r
mean_work_travel_range2 <- seq(
  from = mean_work_travel_min, to = mean_work_travel_max, by = 0.1)
```

Next, we want to feed all these horizontal axis values into `mean_work_travel_ecdf()`, which we do as follows:

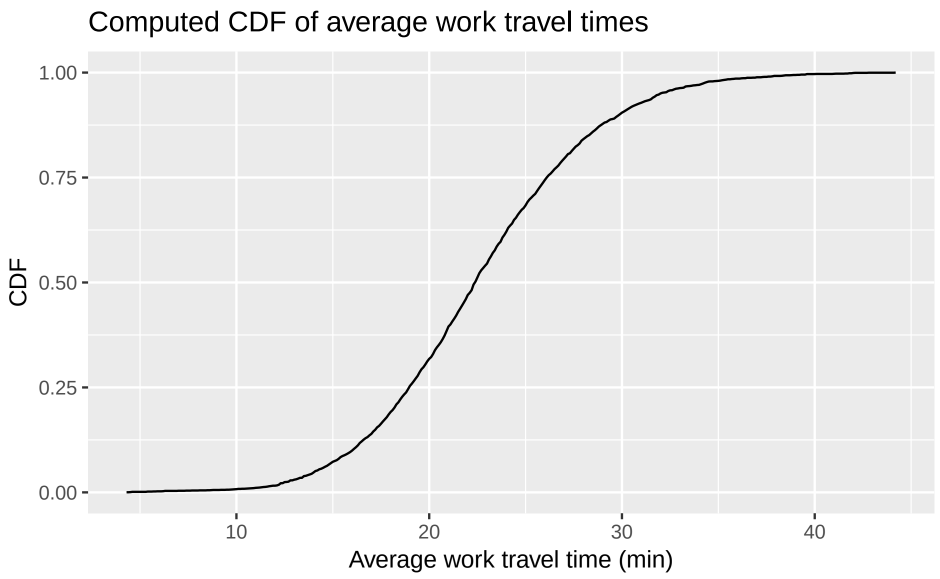
```r
mean_work_travel_computed_cdf <- mean_work_travel_ecdf(mean_work_travel_range2)
```

To create a plot, we combine `mean_work_travel_range2` and `mean_work_travel_computed_cdf` into a `tibble`:

```r
mean_work_travel_cdf_tibble <- data_frame(
  mean_work_travel = mean_work_travel_range2,
  cdf = mean_work_travel_computed_cdf)
```

Now we can visualize it:

```r
mean_work_travel_cdf_tibble %>%
  ggplot() +
  geom_line(mapping = aes(x = mean_work_travel, y = cdf))
```



Now we can easily specify an average work travel time percentile and read the associated time from the plot and vice-versa.

It takes some time to get used to CDFs, but over time it should become clear that they show more information, more clearly, than PMFs.

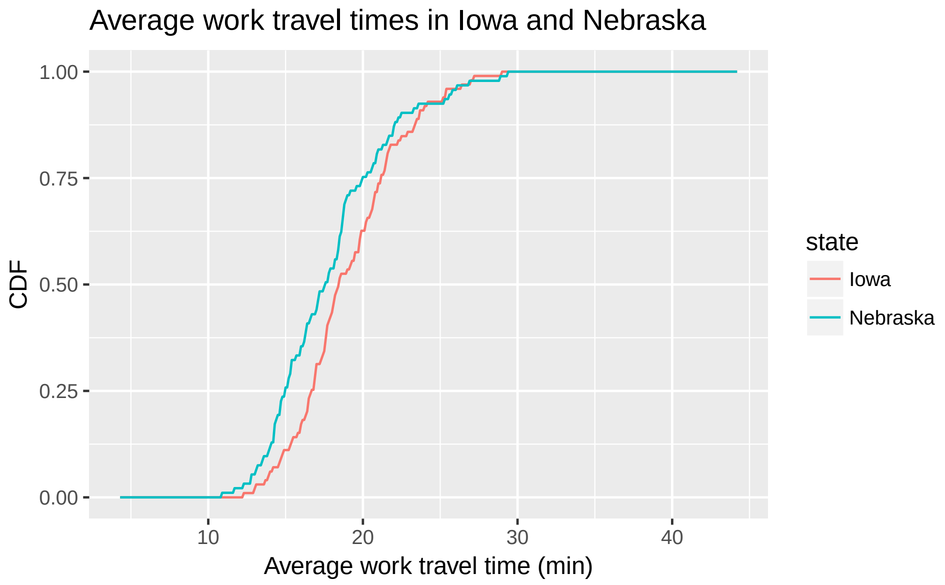
Comparing CDFs

CDFs are especially useful for comparing distributions. Let's revisit the comparison we made between the average work travel times in Nebraska and Iowa. Here is the full code that converts those distributions into CDFs:

```
ia_travel_times_ecdf <- county_complete %>%
  filter(state == "Iowa") %>%
  pull(mean_work_travel) %>%
  ecdf

ne_travel_times_ecdf <- county_complete %>%
  filter(state == "Nebraska") %>%
  pull(mean_work_travel) %>%
  ecdf

mean_work_travel_range <- seq(mean_work_travel_min, mean_work_travel_max, 0.1)

ia_ne_mean_work_travel_cdfs <- data_frame(
  mean_work_travel = mean_work_travel_range,
  cdf_iowa = ia_travel_times_ecdf(mean_work_travel),
  cdf_nebraska = ne_travel_times_ecdf(mean_work_travel)) %>%
  gather(key = state, value = CDF, cdf_iowa:cdf_nebraska) %>%
  mutate(state = recode(state, cdf_iowa = "Iowa", cdf_nebraska = "Nebraska"))
```

and then plots them against each other:

```
ggplot(data = ia_ne_mean_work_travel_cdfs,
       mapping = aes(x = mean_work_travel, y = CDF)) +
  geom_line(mapping = aes(color = state))
```



This visualization makes the shapes of the distributions and the relative differences between them much clearer. We see that Nebraska has shorter average work travel times for most of the distribution, at least until you reach an average time of 25 minutes, after which the Nebraska and Iowa distributions become similar to one another.

Credits