# Univariate Descriptives and Uncertainty

Will Doyle

9/16/2021

## Uncertainty in Univariate Statistics

When we calculate a summary statistic in univariate statistics, we're making a statement about what we can expect to see in other situations. If I say that the average height of a cedar tree is 75 feet, that gives an expectation for the average height we might calculate for any given sample of cedar trees. However, there's more information that we need to communicate. It's not just the summary measure– it's also our level of uncertainty around that summary measure. Sure, the average height might be 75 feet, but does that mean in every sample we ever collect we're always going to see an average of 75 feet?

## Motivation for Today: How much do turnovers matter?

We're going to work with a different dataset covering every NBA game played in the seasons 2016-17 to 2020-21. I'm interested in whether winning teams have higher or lower values of turnovers, and whether winning teams tend to get to the foul line more often.

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --
```

```
## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.6     v dplyr   1.0.7
## v tidyr   1.2.0     v stringr 1.4.0
## v readr   2.1.2     v forcats 0.5.1
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(tidymodels)
```

```
## Registered S3 method overwritten by 'tune':
##   method                   from
##   required_pkgs.model_spec parsnip
```

```
## -- Attaching packages --------------------------------------- tidymodels 0.1.4 --
```

```
## v broom        0.7.11     v rsample      0.1.1
## v dials        0.1.0      v tune         0.1.6
## v infer        1.0.0      v workflows    0.2.4
## v modeldata    0.1.1      v workflowsets 0.1.0
## v parsnip      0.1.7      v yardstick    0.0.9
## v recipes      0.1.17
```

```
## -- Conflicts ------------------------------------------ tidymodels_conflicts() --
## x scales::discard() masks purrr::discard()
## x dplyr::filter()   masks stats::filter()
## x recipes::fixed()  masks stringr::fixed()
```

```
## x dplyr::lag()      masks stats::lag()
## x yardstick::spec() masks readr::spec()
## x recipes::step()   masks stats::step()
## * Search for functions across packages at https://www.tidymodels.org/find/
```

```
library(modelr)
```

```
##
## Attaching package: 'modelr'
```

```
## The following objects are masked from 'package:yardstick':
##
##     mae, mape, rmse
```

```
## The following object is masked from 'package:broom':
##
##     bootstrap
```

## The Data

```
gms<-read_rds("game_summary.Rds")
gms
```

```
## # A tibble: 11,658 x 28
##      idGame yearSeason dateGame   idTeam nameTeam locationGame   tov   pts  treb
##       <dbl>      <int> <date>      <dbl> <chr>    <chr>        <dbl> <dbl> <dbl>
##  1 21600001       2017 2016-10-25 1.61e9 Clevela~ H               14   117    51
##  2 21600001       2017 2016-10-25 1.61e9 New Yor~ A               18    88    42
##  3 21600002       2017 2016-10-25 1.61e9 Portlan~ H               12   113    34
##  4 21600002       2017 2016-10-25 1.61e9 Utah Ja~ A               11   104    31
##  5 21600003       2017 2016-10-25 1.61e9 Golden ~ H               16   100    35
##  6 21600003       2017 2016-10-25 1.61e9 San Ant~ A               13   129    55
##  7 21600004       2017 2016-10-26 1.61e9 Miami H~ A               10   108    52
##  8 21600004       2017 2016-10-26 1.61e9 Orlando~ H               11    96    45
##  9 21600005       2017 2016-10-26 1.61e9 Dallas ~ A               15   121    49
## 10 21600005       2017 2016-10-26 1.61e9 Indiana~ H               16   130    52
## # ... with 11,648 more rows, and 19 more variables: oreb <dbl>, dreb <dbl>,
## #   fga <dbl>, ftm <dbl>, fta <dbl>, pctFG <dbl>, pctFT <dbl>, teamrest <dbl>,
## #   second_game <lgl>, isWin <lgl>, ft_80 <dbl>, game_treb <dbl>,
## #   game_oreb <dbl>, game_dreb <dbl>, opp_treb <dbl>, opp_dreb <dbl>,
## #   opp_oreb <dbl>, oreb_pct <dbl>, tov_pct <dbl>
```

The data for today is game by team summary data. Codebook here. It includes information for each team for every game played from 2017 to 2019.

In his book "Basketball on Paper" Dean Oliver identified the "four factors" to winning basketball:

- Higher Field Goal Percentage
- Lower Turnovers
- Higher Offensive Rebounds
- Get to the foul line frequently

Today we're going to look at turnovers. A turnover in basketball is when a team is on offense, but instead of taking a shot and either making it or missing, the team hands the ball over to the defense. A turnover can occur because of a steal, an intercepted pass, losing the ball, a violation (ie traveling) or an offensive foul.

We're interested in knowing about how turnovers (`tov_pct`) are different between game winners (`isWin`).

## Continuous Variables: Point Estimates

Let's start by calculating average turnovers for winning and losing teams in 2017.

```
tov_pct_summary<-
  gms%>%
  filter(yearSeason==2017)%>%
  group_by(isWin)%>%
  summarize(mean_tov_pct=mean(tov_pct))

tov_pct_summary
```

```
## # A tibble: 2 x 2
##   isWin mean_tov_pct
##   <lgl>        <dbl>
## 1 FALSE        0.146
## 2 TRUE         0.138
```

It looks like there's a fairly substantial difference– winning teams turned the ball over on average in about 13.81 percent of possessions, while losing teams turned it over an average of 14.6 percent of the time.

*Quick Exercise: Calculate average free throw attempts for winning and losing teams in 2017*

What if we take these results and decide that these will apply in other seasons? We could say something like: "Winning teams over the course of a season will turn the ball over t 13.81 percent of the time, and losing teams 14.6 percent of the time, period." Well, let's look and see: did the same result apply in 2018 and 2019?

```
gms%>%
  filter(yearSeason==2018)%>%
  group_by(isWin)%>%
  summarize(mean(tov_pct))
```

```
## # A tibble: 2 x 2
##   isWin `mean(tov_pct)`
##   <lgl>           <dbl>
## 1 FALSE           0.149
## 2 TRUE            0.141
```

```
gms%>%
  filter(yearSeason==2019)%>%
  group_by(isWin)%>%
  summarize(mean(tov_pct))
```

```
## # A tibble: 2 x 2
##   isWin `mean(tov_pct)`
##   <lgl>           <dbl>
## 1 FALSE           0.142
## 2 TRUE            0.134
```

So, no, that's not right. In other seasons winning teams turned the ball over less, but it's not as simple as just saying it will always be the two numbers we calculated from the 2017 data.

What we'd like to be able to do is make a more general statement, not just about a given season but about what we can expect in general. To do that we need to provide some kind of range of uncertainty: what range of turnovers can we expect to see from both winning and losing teams? To generate this range of uncertainty we're going to use some key insights from probability theory and statistics that help us generate estimates of uncertainty.

*Quick exercise: Are winning teams in 2017 more likely to make more free throw attempts than their opponents?*

## Sampling

We're going to start by building up a range of uncertainty from the data we already have. We'll do this by sampling from the data itself. I'll start by using `count` to get the sample size of this data.

```
sample_size<-gms%>%
  filter(yearSeason==2017)%>%
  count()%>%
  as_vector()
```

Whenever we do something that involves randomn numbers, it's a good idea to use `set.seed` to put our random number generators in the same state. This SHOULD mean that we get the same random numbers drawn each time we run the code.

```
set.seed(20210207)
```

Now let's generate a sample from our dataset.

```
gms%>%
  filter(yearSeason==2017)%>% ## Filter to just 2017
  sample_n(size=sample_size, replace = TRUE) %>% ## Sample size is as set above
  group_by(isWin)%>% ## Group by win/lose
  summarize(mean(tov_pct)) ## calculate mean
```

```
## # A tibble: 2 x 2
##   isWin `mean(tov_pct)`
##   <lgl>           <dbl>
## 1 FALSE           0.146
## 2 TRUE            0.136
```

Notice the line: `sample_n(size=sample_size, replace = TRUE)`. That's doing something kind of unusual: it's taking the dataset, then sampling from within it, but creating a new dataset of the same size (`sample_size` is set to be the same size as the data in the lines above). Why in the world would I do this? What this does is to create a sample of data that's based on our own data, but isn't precisely the same. The difference boils down to some elements if the data being repeated.

```
gms%>%
  filter(yearSeason==2017)%>%
  sample_n(size=sample_size, replace = TRUE)%>%
  group_by(idGame,nameTeam)%>%
  mutate(replicates=n())%>%
  arrange(-replicates,idGame,nameTeam)%>%
  select(idGame,yearSeason,dateGame,nameTeam,replicates)
```

```
## # A tibble: 2,460 x 5
## # Groups:   idGame, nameTeam [1,546]
##      idGame yearSeason dateGame   nameTeam          replicates
##       <dbl>      <int> <date>     <chr>                  <int>
##  1 21600849       2017 2017-02-16 Chicago Bulls              6
##  2 21600849       2017 2017-02-16 Chicago Bulls              6
##  3 21600849       2017 2017-02-16 Chicago Bulls              6
##  4 21600849       2017 2017-02-16 Chicago Bulls              6
##  5 21600849       2017 2017-02-16 Chicago Bulls              6
##  6 21600849       2017 2017-02-16 Chicago Bulls              6
##  7 21600361       2017 2016-12-12 Charlotte Hornets          5
##  8 21600361       2017 2016-12-12 Charlotte Hornets          5
##  9 21600361       2017 2016-12-12 Charlotte Hornets          5
## 10 21600361       2017 2016-12-12 Charlotte Hornets          5
```

```
## # ... with 2,450 more rows
```

See how some games/teams are repeated multiple times?

Let's do that again and see what it does to our summary numbers:

```
gms%>%
  filter(yearSeason==2017)%>% ## Filter to just 2017
  sample_n(size=sample_size,replace = TRUE) %>% ## Sample size is as set above
  group_by(isWin)%>% ## Group by win/lose
  summarize(mean(tov_pct)) ## calculate mean
```

```
## # A tibble: 2 x 2
##   isWin `mean(tov_pct)`
##   <lgl>           <dbl>
## 1 FALSE           0.145
## 2 TRUE            0.138
```

It changes a bit each time, dependent on which random resample (or replicate) we use. These replicates are called bootstrap replicates, and we can use specific code to generate them.

```
gms%>%
  filter(yearSeason==2017)%>%
  resample_bootstrap()%>%
  as_tibble()%>%
  group_by(isWin)%>% ## Group by win/lose
  summarize(mean(tov_pct)) ## calculate mea
```

```
## # A tibble: 2 x 2
##   isWin `mean(tov_pct)`
##   <lgl>           <dbl>
## 1 FALSE           0.144
## 2 TRUE            0.136
```

I can continue this process of sampling and generating values many times using a loop. The code below resamples from the data 10,000 times, each time calculating the mean turnovers for winners and losers in a sample of size 100. It then adds those two means to a growing list, using the bind_rows function.

### Warning: the code below will take a little while to run

```
gms_tov_rs<-NULL ##  Create a NULL variable: will fill this in later

for (i in 1:1000){ # Repeat the steps below 10,000 times
  gms_tov_rs<-gms%>% ## Create a dataset called gms_tov_rs (rs=resampled)
  filter(yearSeason==2017)%>%  ## Just 2017
  resample_bootstrap()%>%
  as_tibble()%>%
  group_by(isWin)%>% ## Group by won or lost
  summarize(mean_tov_pct=mean(tov_pct))%>% ## Calculate mean turnovers for winners and losers
    bind_rows(gms_tov_rs) ## add this result to the existing dataset
}
```

Now I have a dataset that is built up from a bunch of resamples from the data, with average turnovers for winners and losers in each sample. Let's see what these look like.

```
gms_tov_rs
```

```
## # A tibble: 2,000 x 2
```

```
##    isWin mean_tov_pct
##    <lgl>        <dbl>
##  1 FALSE        0.146
##  2 TRUE         0.137
##  3 FALSE        0.147
##  4 TRUE         0.137
##  5 FALSE        0.146
##  6 TRUE         0.139
##  7 FALSE        0.145
##  8 TRUE         0.136
##  9 FALSE        0.146
## 10 TRUE         0.136
## # ... with 1,990 more rows
```

This is a dataset that's just a bunch of means. We can calculate the mean of all of these means and see what it looks like:

```
gms_tov_rs%>%
  group_by(isWin)%>%
  summarise(mean_of_means=mean(mean_tov_pct))
```

```
## # A tibble: 2 x 2
##   isWin mean_of_means
##   <lgl>         <dbl>
## 1 FALSE         0.146
## 2 TRUE          0.138
```

How does this "mean of means" compare with the actual?

```
gms%>%
  filter(yearSeason==2017)%>%
  group_by(isWin)%>%
  summarize(mean(tov_pct))
```

```
## # A tibble: 2 x 2
##   isWin `mean(tov_pct)`
##   <lgl>           <dbl>
## 1 FALSE           0.146
## 2 TRUE            0.138
```

Pretty similar! It's what we would expect, really, but it's super important. If we repeatedly sample from a dataset, our summary measures of a sufficiently large number of repeated samples will converge on the true value of the measure from the dataset.

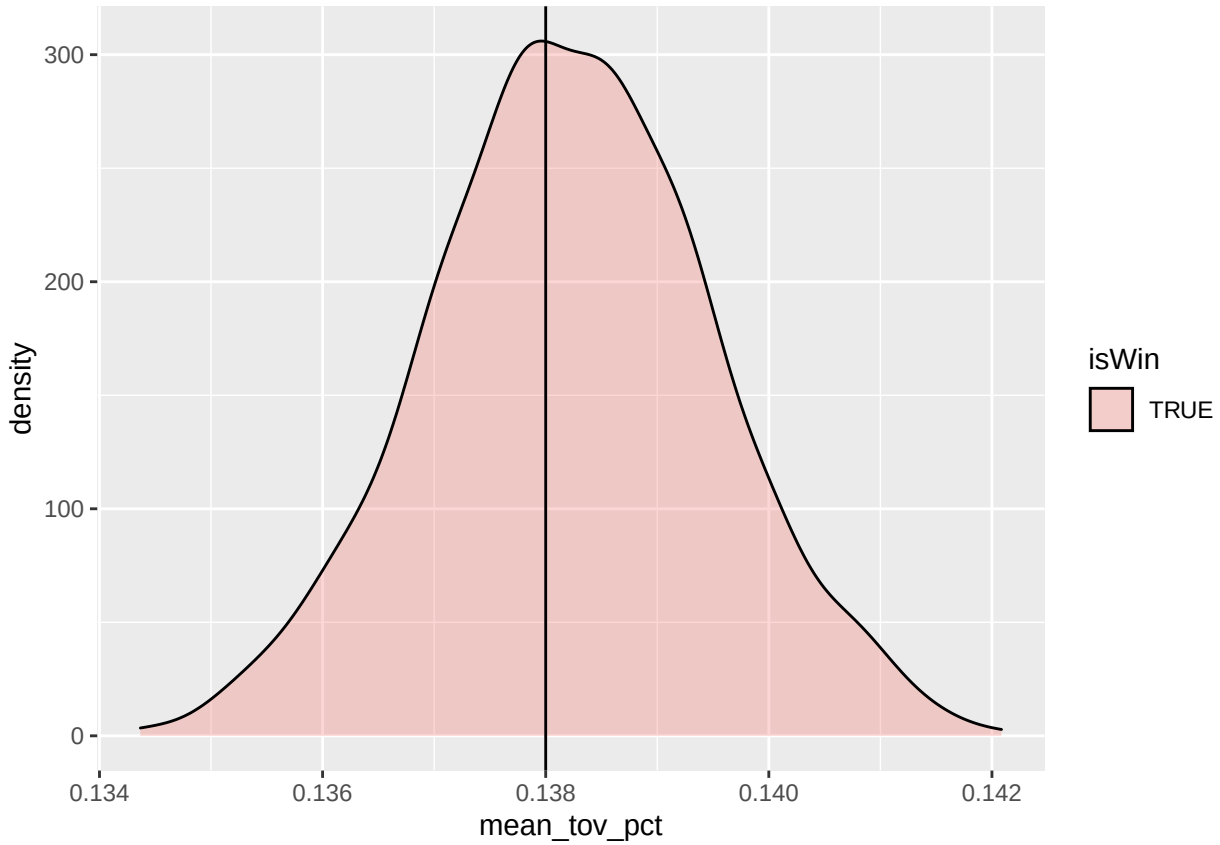
*Quick Exercise: Repeat the above, but do it for free throw attempts.

## Distribution of Resampled Means

That's fine, but the other thing is that the *distribution* of those repeated samples will tell us about what we can expect to see in other, out of sample data that's generated by the same process.

Let's take a look at the distribution of turnovers for game winners:

```
gms_tov_rs%>%
  filter(isWin)%>%
  ggplot(aes(x=mean_tov_pct,fill=isWin))+
  geom_density(alpha=.3)+
  geom_vline(xintercept =.138)
```




We can see that the mean of this distribution is centered right on the mean of the actual data, and it goes from about 11 to about 15. This is different than the minimum and maximum of the overall sample, which goes from 0.134366248883547 to 0.134366248883547.

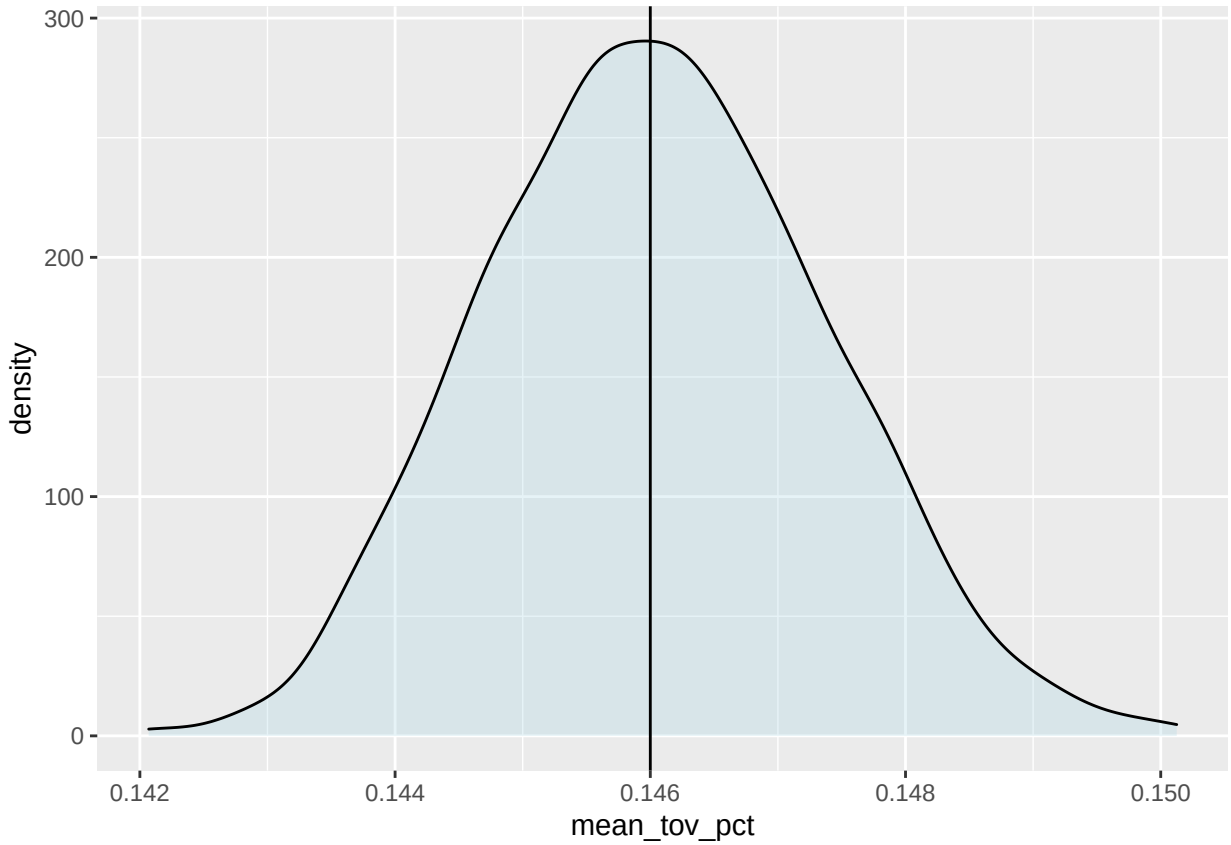
We can use the function `fivenum` to provide a summary of this distribution.

```
gms_tov_rs%>%
  filter(isWin)%>%
  summarize(value=fivenum(mean_tov_pct))%>% ## Five number summary: described below
  mutate(measure=c("Min","25th percentile","Median","75th percentile","Max"))%>%
  select(measure, value)
```

```
## # A tibble: 5 x 2
##   measure         value
##   <chr>           <dbl>
## 1 Min             0.134
## 2 25th percentile 0.137
## 3 Median          0.138
## 4 75th percentile 0.139
## 5 Max             0.142
```

And for game losers:

```
gms_tov_rs%>%
  filter(!isWin)%>%
  ggplot(aes(x=mean_tov_pct,fill=isWin))+
  geom_density(alpha=.3,fill="lightblue")+
  geom_vline(xintercept =.146)
```




```
gms_tov_rs%>%
  filter(!isWin)%>%
  summarize(value=fivenum(mean_tov_pct))%>%
    mutate(measure=c("Min","25th percentile","Median","75th percentile","Max"))%>%
  select(measure, value)
```

```
## # A tibble: 5 x 2
##   measure         value
##   <chr>           <dbl>
## 1 Min             0.142
## 2 25th percentile 0.145
## 3 Median          0.146
## 4 75th percentile 0.147
## 5 Max             0.150
```

*Quick Exercise: Calculate the same summary, but do it for free throw attempts.*

## So What? Using Percentiles of the Resampled Distribution

Now we can make some statements about uncertainty. Based on this what we can say is that in other seasons, we would expect that turnover for game winners will be in a certain range, and the same for game losers. What range? Well it depends on the level of risk you're willing to take as an analyst. Academics (a cautious bunch to be sure) usually use the middle 95 percent of the distribution: So for game winners:

```
gms_tov_rs%>%
  filter(isWin)%>%
  summarize(pct_025=quantile(mean_tov_pct,.025),
            pct_975=quantile(mean_tov_pct,.975))
```

```
## # A tibble: 1 x 2
##   pct_025 pct_975
##     <dbl>   <dbl>
## 1   0.136   0.141
```

This tells us we can expect that game winners in future seasons will turn the ball over in the range specified above.

What about for game losers? How often will they be expected to turn the ball over?

```
gms_tov_rs%>%
group_by(isWin)%>%
  summarize(pct_025=quantile(mean_tov_pct,.025),
            pct_975=quantile(mean_tov_pct,.975))
```

```
## # A tibble: 2 x 3
##   isWin pct_025 pct_975
##   <lgl>   <dbl>   <dbl>
## 1 FALSE   0.144   0.149
## 2 TRUE    0.136   0.141
```

Let's check to see if our expectations are borne out in future seasons, by calculating the actual values of turnover percentage for those subseqeuent seasons.

```
gms%>%
  filter(yearSeason==2018)%>%
  group_by(isWin)%>%
  summarize(mean(tov_pct))
```

```
## # A tibble: 2 x 2
##   isWin `mean(tov_pct)`
##   <lgl>           <dbl>
## 1 FALSE           0.149
## 2 TRUE            0.141
```

```
gms%>%
  filter(yearSeason==2019)%>%
  group_by(isWin)%>%
  summarize(mean(tov_pct))
```

```
## # A tibble: 2 x 2
##   isWin `mean(tov_pct)`
##   <lgl>           <dbl>
## 1 FALSE           0.142
## 2 TRUE            0.134
```

While our ranges are close, they're not exact– we can see values outside the range for subsequent seasons.

## Other intervals– the tradeoff between a "precise" interval and risk

You may be underwhelmed at this point, because the 95 percent range is a big range of possible turnover values. We can use narrower intervals– it just raises the risk of being wrong. Let's try the middle 50 percent.

```
gms_tov_rs%>%
  group_by(isWin)%>%
  summarize(pct_25=quantile(mean_tov_pct,.25),
            pct_75=quantile(mean_tov_pct,.75))
```

```
## # A tibble: 2 x 3
```

```
##   isWin pct_25 pct_75
##   <lgl>  <dbl>  <dbl>
## 1 FALSE  0.145  0.147
## 2 TRUE   0.137  0.139
```

Now let's compare that middle 50% with the actual values in 2018 and 2019.

```
gms%>%
  filter(yearSeason==2018)%>%
  group_by(isWin)%>%
  summarize(mean(tov_pct))
```

```
## # A tibble: 2 x 2
##   isWin `mean(tov_pct)`
##   <lgl>           <dbl>
## 1 FALSE           0.149
## 2 TRUE            0.141
```

```
gms%>%
  filter(yearSeason==2019)%>%
  group_by(isWin)%>%
  summarize(mean(tov_pct))
```

```
## # A tibble: 2 x 2
##   isWin `mean(tov_pct)`
##   <lgl>           <dbl>
## 1 FALSE           0.142
## 2 TRUE            0.134
```

Let's just keep going and try the middle 10%, and compare that with the values in 2018 and 2019.

```
gms_tov_rs%>%
  group_by(isWin)%>%
  summarize(pct_45=quantile(mean_tov_pct,.45),
            pct_55=quantile(mean_tov_pct,.55))
```

```
## # A tibble: 2 x 3
##   isWin pct_45 pct_55
##   <lgl>  <dbl>  <dbl>
## 1 FALSE  0.146  0.146
## 2 TRUE   0.138  0.138
```

```
gms%>%
  filter(yearSeason==2018)%>%
  group_by(isWin)%>%
  summarize(mean(tov))
```

```
## # A tibble: 2 x 2
##   isWin `mean(tov)`
##   <lgl>       <dbl>
## 1 FALSE        14.1
## 2 TRUE         13.3
```

```
gms%>%
  filter(yearSeason==2019)%>%
  group_by(isWin)%>%
  summarize(mean(tov))
```

```
## # A tibble: 2 x 2
```

```
##   isWin `mean(tov)`
##   <lgl>       <dbl>
## 1 FALSE        13.9
## 2 TRUE         13.1
```

It turns out that the way this method works is that for an interval of a certain range, the calculated interval will include the true value of the measure in the same percent *of repeated samples*. We can think of eac h season as a repeated sample, so the middle 95 percent of this range will include the true value in 95 percent of seasons. When we call this a confidence interval, we're saying we have confidence in the approach, not the particular values we calculated.

The tradeoff here is between providing a narrow range of values vs. the probability of being correct. We can give a very narrow interval for what we would expect to see in out of sample data, but we're going to be wrong– a lot. We can give a very wide interval, but the information isn't going to be useful to decisionmakers. This is one of the key tradeoffs in applied data analysis, and there's no single answer to the question: what interval should I use? Academic work has settled on the 95 percent interval, but there's no real theoretical justification for this.

## Empirical Bootstrap

What we just did is called the empirical bootstrap. It's massively useful, because it can be applied for any summary measure of the data: median, percentiles, and measures like regression coefficients. Here is the summary of steps for the empirical bootstrap:

- Decide on the summary measure to be used for the variable (it doesn't have to be the mean)
- Calculate the summary measure on a resampled version of the data– this is called the bootstrap replicate.
- Repeat step 2 many times (how many? Start with 1000, but more is better.) Compile the estimates.
- Calculate the percentiles of the bootstrap distribution from the previous step.
- Describe your uncertainty using those percentiles. d

*Quick Exercise: Does the 95 percent interval for free throws attempted for winning and losing teams include the values for subsequent seasons?*

## What's wrong with this particular application

The idea here is that the bootstrap confidence interval applies to data drawn from the *exact same* data generating process. Usually that's in reference to new samples drawn from an infinitely large population. In this case, it's just subsequent seasons. But the NBA changes! In particular, the rules and the way the rules are enforced change over time, so it's not quite accurate to think of this as the exact same data generating process. Even still, in the absence of another way to think about uncertainty, the empirical bootstrap gives us a really good way to quantify how uncertain we really are.

## Appendix: Calculating Bootstraps Using Rsample

We can undertake the steps above using R's built-in capabilities. Below I create a dataset that's structured for bootstrap resampling:

```
boot_2017<-bootstraps(gms%>%filter(yearSeason==2017),times = 10000)
```

This is what's called a "splits" data structure. It splits the data into two parts: one part will be used in the analysis, one part will be held out. For reasons that escape me, the command only allows the data to be split 90/10, with 90 percent held out for analysis and 10 percent for assessment.

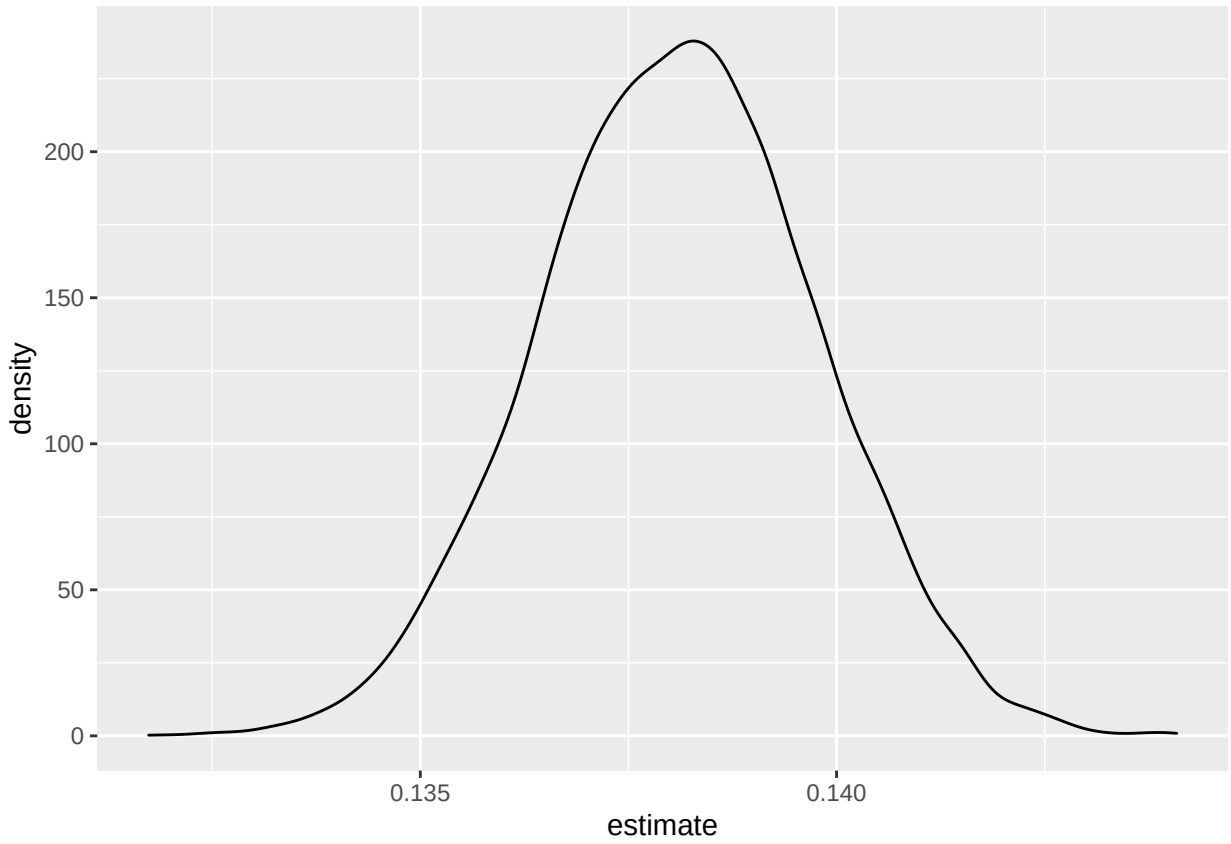
The function below takes the data (in split format), samples each element down to the specified sample size (100 in our case) and then pulls the turnover variable `tov`. It then returns a dataset that includes just the mean of the specified variable, in this case `tov`.

```r
calc_tov_mean_winners <- function(split){
  dat <- assessment(split) %>% ## create an object called dat from each "split" of the data
    filter(isWin)%>% ## filter just for winners
    pull(tov_pct) ## pull just turnovers

  # Put it in this tidy format to use int_pctl
  return(tibble( ## return a tibble
    term = "mean", ## the variable will be named mean
    estimate = mean(dat))) ## the estimate is the mean of dat from above
}

calc_tov_mean_losers <- function(split){
  dat <- assessment(split) %>% ## create an object called dat from each "split" of the data
    filter(!isWin)%>% ## filter just for losers
    pull(tov_pct) ## pull just turnovers

  # Put it in this tidy format to use int_pctl
  return(tibble( ## return a tibble
    term = "mean", ## the variable will be named mean
    estimate = mean(dat))) ## the estimate is the mean of dat from above
}
```

```r
results_winners<-boot_2017%>% ## start with the resampled dataset
  mutate(tov_mean= ## mutate to create a column called tov_mean
           map(splits,calc_tov_mean_winners))  ## map the "calc" function onto each split

results_winners%>%
  int_pctl(tov_mean)
```

```
## # A tibble: 1 x 6
##   term  .lower .estimate .upper .alpha .method
##   <chr>  <dbl>     <dbl>  <dbl>  <dbl> <chr>
## 1 mean   0.135     0.138  0.141   0.05 percentile
```

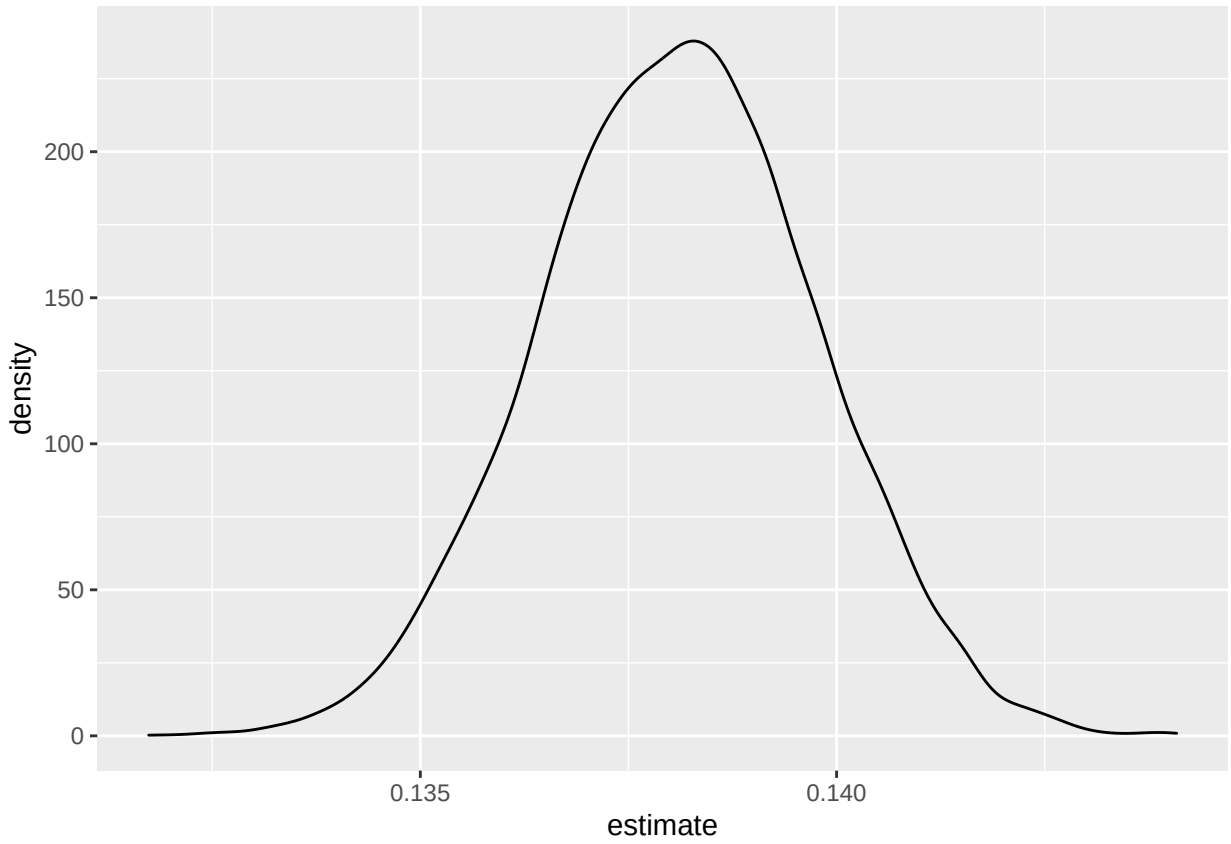
```r
results_winners%>%
  select(tov_mean)%>%
  unnest(cols=tov_mean)%>%
  ggplot(aes(x=estimate))+
  geom_density()
```




```
results_losers<-boot_2017%>% ## start with the resampled dataset
  mutate(tov_mean= ## mutate to create a column called tov_mean
           map(splits,calc_tov_mean_losers))  ## map the "calc" function onto each split

results_losers%>%int_pctl(tov_mean)
```

```
## # A tibble: 1 x 6
##   term  .lower .estimate .upper .alpha .method
##   <chr>  <dbl>     <dbl>  <dbl>  <dbl> <chr>
## 1 mean   0.143     0.146  0.149   0.05 percentile
```

```
results_winners%>%
  select(tov_mean)%>%
  unnest(cols=tov_mean)%>%
  ggplot(aes(x=estimate))+
  geom_density()
```


200
150
100
50
0
density
0.135
0.140
estimate