Lecture 8: More Simulations

STAT GR5206 *Statistical Computing & Introduction to Data Science*

Gabriel Young
Columbia University

June 19, 2017

Last Time

- **Pseudo-random Number Generators**. What makes numbers 'random' and the Linear Congruential Estimator.
- **Simulating Random Variables in** `R`: Built-in `R` functions for common distributions and the `sample()` function.
- **Simulating Random Variables from Uncommon Distributions**. Use the Accept-Reject Algorithm.

This Time

- **Permutation Tests**. Are two distributions the same?
- **More Simulations**.

Permutation Tests

Cats

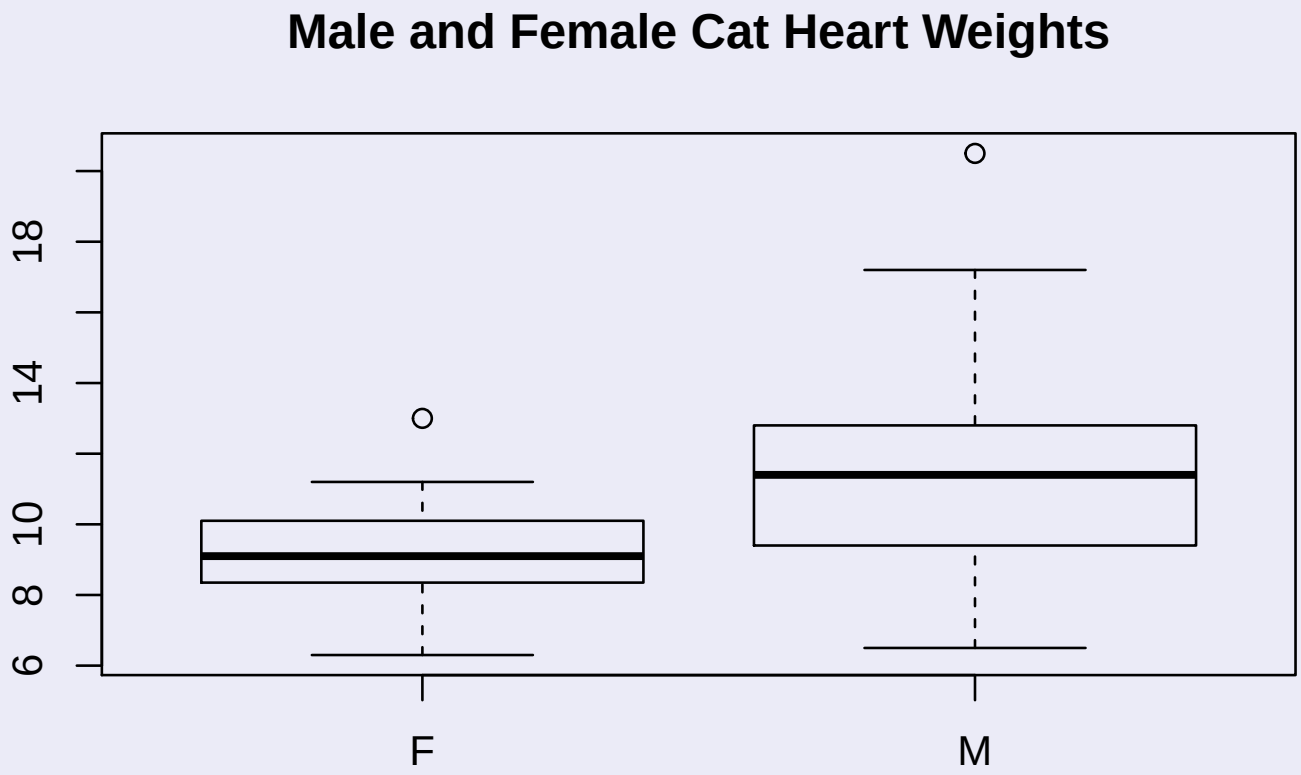
The cats dataset includes the heart and body weights of samples of male and female cats. All the cats are adults and over 2 kg in body weight.

```
> # install.packages("MASS")
> library(MASS)
> head(cats)

  Sex Bwt Hwt
1   F 2.0 7.0
2   F 2.0 7.4
3   F 2.0 9.5
4   F 2.1 7.2
5   F 2.1 7.3
6   F 2.1 7.6
```

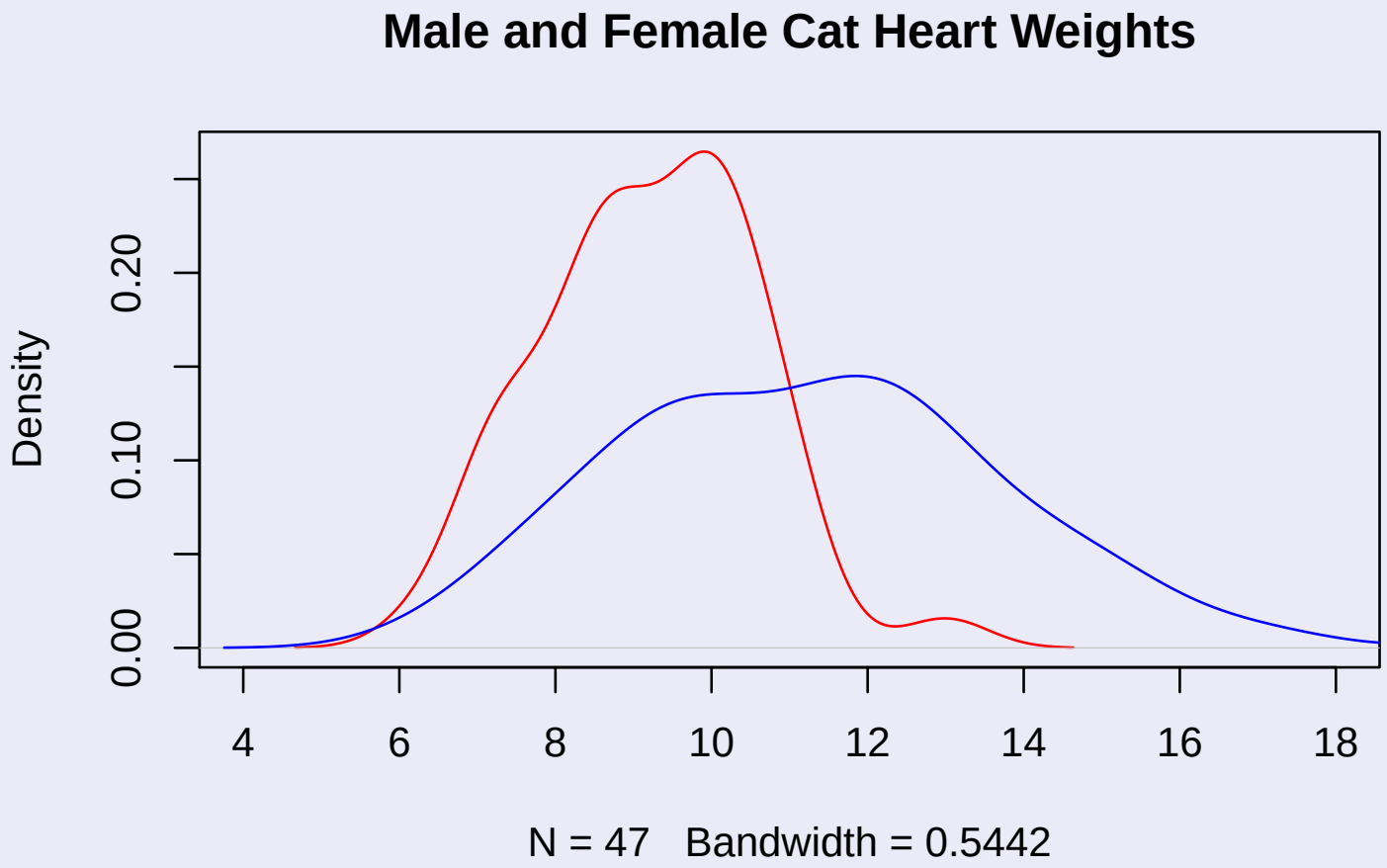
Male and Female Cat Heart Weights

```
> boxplot(cats$Hwt ~ cats$Sex,
+         main = "Male and Female Cat Heart Weights")
```



Male and Female Cat Heart Weights

```
> plot(density(cats$Hwt[cats$Sex == "F"]), col = "red",
+      xlim = c(4, 18), main = "Male and Female Cat Heart Weights")
> lines(density(cats$Hwt[cats$Sex == "M"]), col = "blue")
```



Male and Female Cat Heart Weights

Recall the Independent Two-Sample T-test

- Tests if the population means of two samples are equal. $H_0: \mu_{male} - \mu_{female} = 0$
- Assume the data in each sample come from a normal distribution.

$H_1: \mu_{male} - \mu_{female} \neq 0$

```
> girlcats <- cats$Sex == "F"
> t.test(cats$Hwt[girlcats], cats$Hwt[!girlcats])

Welch Two Sample t-test

data:  cats$Hwt[girlcats] and cats$Hwt[!girlcats]
t = -6.5179, df = 140.608, p-value = 1.186e-09
alternative hypothesis: true difference in means is not equal
95 percent confidence interval:
 -2.763753 -1.477352
sample estimates:
mean of x mean of y
 9.202128 11.322680
```

test stat $t = \dfrac{\bar{x}_m - \bar{x}_F - 0}{\sqrt{\frac{s^2_m}{n_m} + \frac{s^2_F}{n_F}}}$

→ heart weight is different for male & female

Male and Female Cat Heart Weights

What if the assumptions don't hold?

- In our dataset, 47 female cats and 97 male cats.
- We study test statistic:

$$\hat{D} = mean(X_F) - mean(X_M),$$

where X_F are female cat heart weights and X_M are male cat heart weights.

- For our data $\hat{D} = -2.12$.

$$t = \frac{-2.12}{\sqrt{\frac{S_M^2}{n_M} + \frac{S_F^2}{n_F}}}$$

↓

is this significant not assuming normality?

Male and Female Cat Heart Weights

What if the assumptions don't hold?

- In our dataset, 47 female cats and 97 male cats.
- We study test statistic:

$$\hat{D} = mean(X_F) - mean(X_M),$$

where X_F are female cat heart weights and X_M are male cat heart weights.

- For our data $\hat{D} = -2.12$.

Permutation Principle

If there were no difference in male and female heart weights (null hypothesis), then all datasets obtained by randomly assigning 47 of the values in `cats$Hwt` to female cats and 97 to male cats would have equal chance of being observed in the study.

Male and Female Cat Heart Weights

What if the assumptions don't hold?

This gives us

$$\binom{144}{47} = \frac{144!}{47!\,97!} = 2.231033 \times 10^{38}$$

possible two-sample datasets (under the null hypothesis).

How rare is our result?

How many of these datasets have $|mean(X_F) - mean(X_M| \geq 2.12$? What is the probability of seeing differences in the group means as extreme (or more extreme) than ours?

Male and Female Cat Heart Weights

What if the assumptions don't hold?

- Could consider each of the 2.231033×10^{38} possible two-sample datasets, calculate $|mean(X_F) - mean(X_M)|$ for each, and then compare to our original $\hat{D}$...
- Instead we use a **permutation test**, which randomly samples from the 2.231033×10^{38} possible two-sample datasets and estimates a p-value based on our original $\hat{D}$.

Permutation Test

Steps

For step $i = 1, 2, \ldots, P$,

- Randomly assign heart weights in `cats$Hwt` as either male or female with exactly 97 males and 47 females.
- Compute $\hat{D}_i = |mean(X_{F,i}) - mean(X_{M,i})|$ where $X_{F,i}$ are female cat heart weights in step i and $X_{M,i}$ are male cat heart weights.

Finally, we calculate a (two-sided) p-value as follows:

$$\frac{1}{P} \sum_{i=1}^{P} \mathbb{I}(|\hat{D}_i| \geq |\hat{D}|).$$

if this value is small it supports that there is a statiscal difference.

Permutation Test

What are we actually testing?

- Using the permutation test, we test the null hypothesis that the two samples are from the same distribution.
- How do we do this? Have we seen enough evidence (in the form of the observed difference between the sample means being large enough) to reject the null hypothesis that the two groups have identical probability distributions?

Check Yourself: Permutation Test

Task

Fill in the following code to run a permutation test on the `cats$Hwt` data.

```
> girlcats <- cats$Sex == "F"
> Dhat     <- mean(cats$Hwt[girlcats])
>                - mean(cats$Hwt[!girlcats])
> nf   <- sum(girlcats); nm <- sum(!girlcats)
> P    <- 10000
> sample_diffs <- rep(NA, P)
> for (i in 1:P) {
+   ####################
+   ## Add code here ##
+   ####################
+ }
> pval <- mean(abs(sample_diffs) >= abs(Dhat))
```

Check Yourself: Permutation Test

Solution

```
> girlcats <- cats$Sex == "F"
> Dhat <- mean(cats$Hwt[girlcats])-mean(cats$Hwt[!girlcats])
> nf   <- sum(girlcats); nm <- sum(!girlcats)
> P    <- 10000
> sample_diffs <- rep(NA, P)
> for (i in 1:P) {
+   perm_data <- cats$Hwt[sample(1:(nf+nm))]
+   meanf     <- mean(perm_data[1:nf])
+   meanm     <- mean(perm_data[-(1:nf)])
+   sample_diffs[i] <- meanf - meanm
+ }
> pval <- mean(abs(sample_diffs) >= abs(Dhat))
> pval
[1] 0
```

→ female

→ male

→ 0 occurance where random difference did not exceed real difference

More Simulations

Random Number Generation in R

Recall,

We can simulate random numbers in R from various distributions:

- `rnorm()`: generate normal random variables
- `pnorm()`: normal distribution function, $\Phi(x) = P(Z < x)$
- `dnorm()`: normal density function, $\phi(x) = \Phi'(x)$
- `qnorm()`: normal quantile function, $\Phi^{-1}(u)$

Random Number Generation in R

Recall,

We can simulate random numbers in R from various distributions:

- `rnorm()`: generate normal random variables
- `pnorm()`: normal distribution function, $\Phi(x) = P(Z < x)$
- `dnorm()`: normal density function, $\phi(x) = \Phi'(x)$
- `qnorm()`: normal quantile function, $\Phi^{-1}(u)$

Replace `norm` with other distribution names, all the same functions apply.

Why do we Simulate?

Recall,

R gives us great simulation tools (unlike many other languages). Why should we simulate, though?

- Often simulations are easier than hand calculations.
- Often simulations can be made more realistic than hand calculations.

Today We'll do a FUN Simulation! [2]

Darts

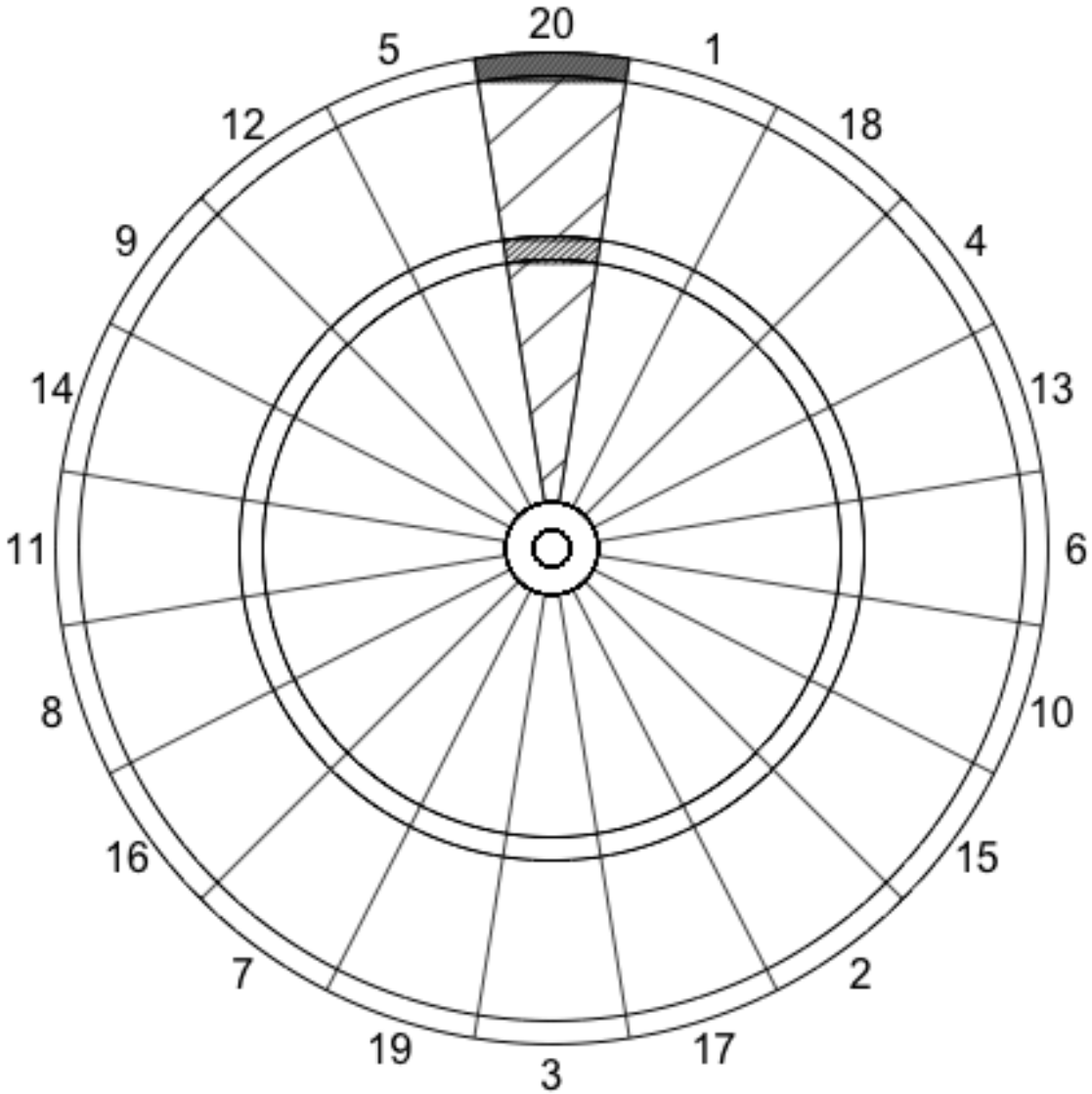
Darts is a game where you throw metal darts at a dartboard and receive different scores for landing the dart in different regions.

- Land in a slice, obtain the score of the corresponding number.
- Land in the "double ring", get double the score.
- Land in the "triple ring", get triple the score.
- Land in the "double bullseye", get 50 points.
- Land in the "single bullseye", get 25 points.

1

[1] Darts simulation notes adapted from http://www.stat.cmu.edu/ ryantibs/darts/

Today We'll do a FUN Simulation!



Some Questions to Answer with Simulation

- What strategy provides the better score? Throwing such that darts land uniformly at random on the board, or aiming at the center, with your throws being normally distributed with some variance.
- If you're aiming at a spot and your throws are normally distributed with some variance, what spot should you aim for?
- Can I estimate the variance of my throws, if I aim at the center and receive a certain score?

Let's Simulate Dart Throws

Board Measurements

We'll make a list of standard dart board measurements and the numbers of the board. All measurements are in mm.

```
> board = list(
+   R1 = 6.35,  # center to double bullseye ring
+   R2 = 15.9,  # center to single bullseye ring
+   R3 = 99,    # center to inner triple ring
+   R4 = 107,   # center to outer triple ring
+   R5 = 162,   # center to inner double ring
+   R = 170,    # center to outer double ring
+   nums = c(20,1,18,4,13,6,10,15,2,17,3,19,
+            7,16,8,11,14,9,12,5)) # numbers in order
```

Let's Simulate Dart Throws

Plotting the Board

We'll write a `drawBoard` function with the following features:

- Inputs: `board` a list containing dart board measurements.
- Outputs: None, but it will plot the board. Then we can plot on top of the board with functions like `points()` or `lines()`.

Let's Simulate Dart Throws

Plotting the Board

We'll write a `drawBoard` function with the following features:

- Inputs: `board` a list containing dart board measurements.
- Outputs: None, but it will plot the board. Then we can plot on top of the board with functions like `points()` or `lines()`.

This function is in the student script and it essentially does the following:

- Use $(0, 0)$ as the center of the board.
- Plot concentric circles centered at $(0, 0)$ to show the rings.
- Plot the 'spokes'.
- Add the numbers around the outside.

Let's Simulate Dart Throws

Code Example

Let's Simulate Dart Throws

Simulating the Scoring

We write a `scorePositions()` function with the following features.

- Inputs:
  - `x`: vector, horizontal positions of dart throws in mm, with the center of the board being 0
  - `y`: vector, vertical positions of dart throws in mm, with the center of the board being 0
  - `board`: list, containing dart board measurements
- Outputs: vector of scores, same length as x (and as y)

Let's Simulate Dart Throws

Simulating the Scoring

We write a `scorePositions()` function with the following features.

- Inputs:
  - `x`: vector, horizontal positions of dart throws in mm, with the center of the board being 0
  - `y`: vector, vertical positions of dart throws in mm, with the center of the board being 0
  - `board`: list, containing dart board measurements
- Outputs: vector of scores, same length as x (and as y)

This function is in the student script and it essentially does the following:

- Calculate the radius at which the throw lands.
- Calculate the angle at which the throw lands.
- Use these to calculate the score.

Let's Simulate Dart Throws

Code Example

Let's Simulate Dart Throws

Our Task

Let `x`, `y` denote the horizontal and vertical positions of the throws, measured from the center of the board which is $(0, 0)$. Let's consider the following model, `x` and `y` are both $N(0, 50^2)$ (the standard deviation here is 50mm). We will:

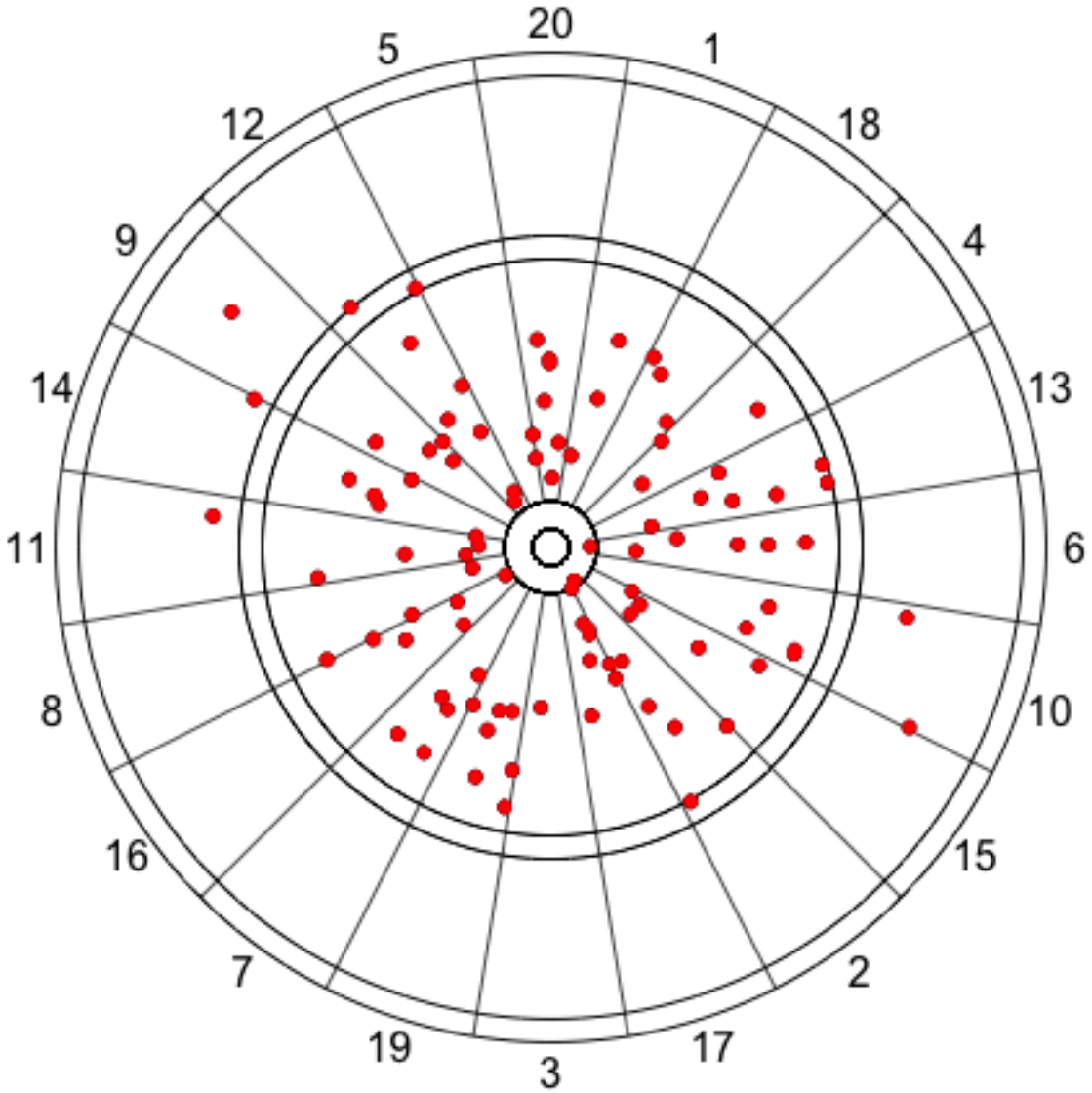
- Generate 100 throws from this model.
- Plot the throw positions on the dart board, with `pch` $= 20$ (to make small solid dots).
- Compute the scores of the 100 throws and add the scores corresponding to each throw on the plot.

Let's Simulate Dart Throws

Our Solution

```
> # We simulate 100 throws with the x and y
> # location modeled by N(0, 50^2).
> throws  <- 100
> std.dev <- 50
> x <- rnorm(throws, sd = std.dev)
> y <- rnorm(throws, sd = std.dev)
> drawBoard(board)
> points(x, y, pch = 20, col = "red")
```

Let's Simulate Dart Throws

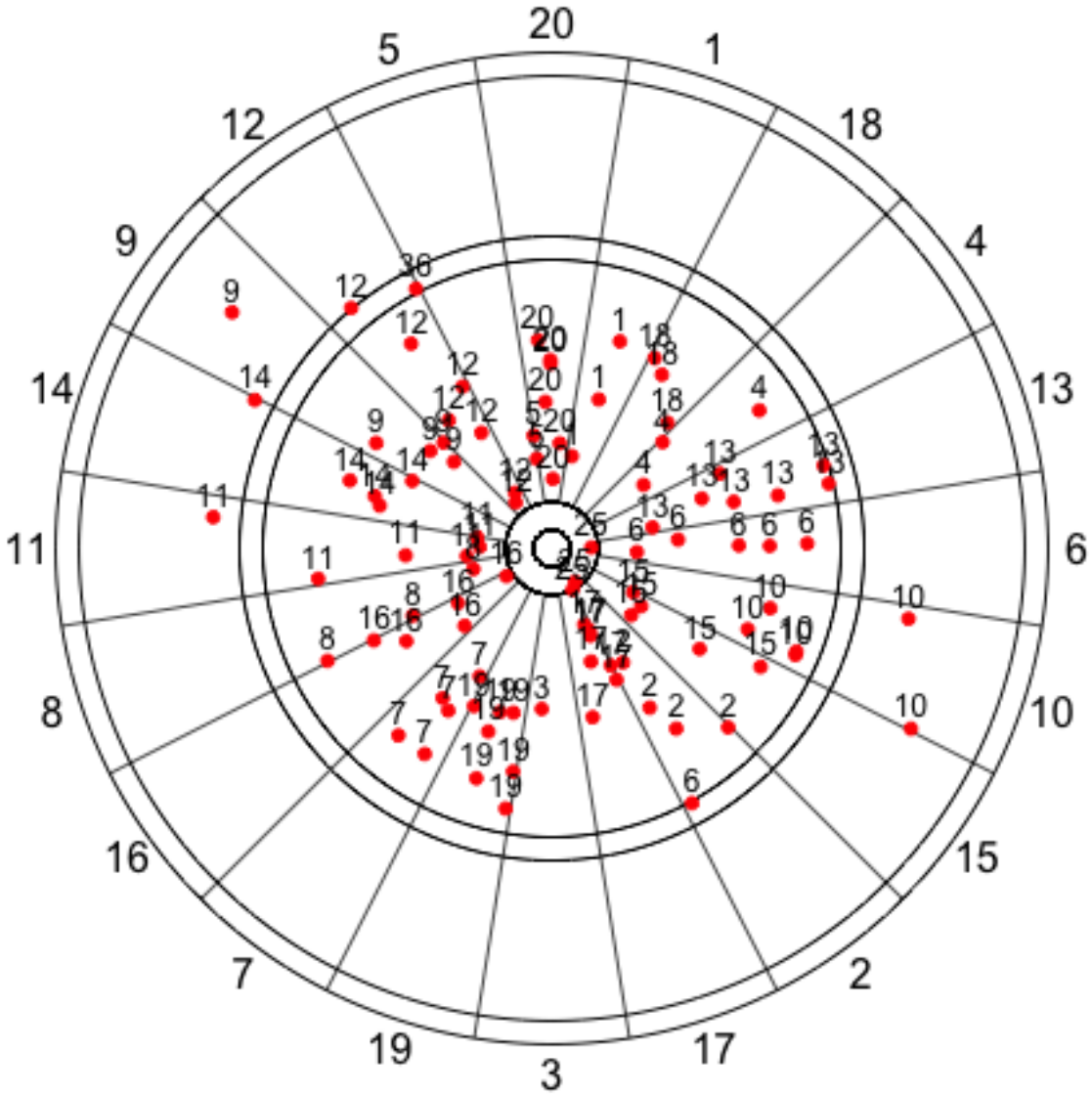


Let's Simulate Dart Throws

Our Solution

```
> # We score our throws and add the values to the plot
> scores <- scorePositions(x, y, board)
> text(x, y + 8, scores, cex = .75)
```

Let's Simulate Dart Throws



Check Yourself

Task

Let `x`,`y` denote the horizontal and vertical positions of the throws, measured from the center of the board which is $(0, 0)$. Consider the following model, `x` and `y` are both $Uniform(-R, R)$ where $R = 170$ is the radius of the board. Note this is the smallest rectangle that contains the dart board. Do the following:

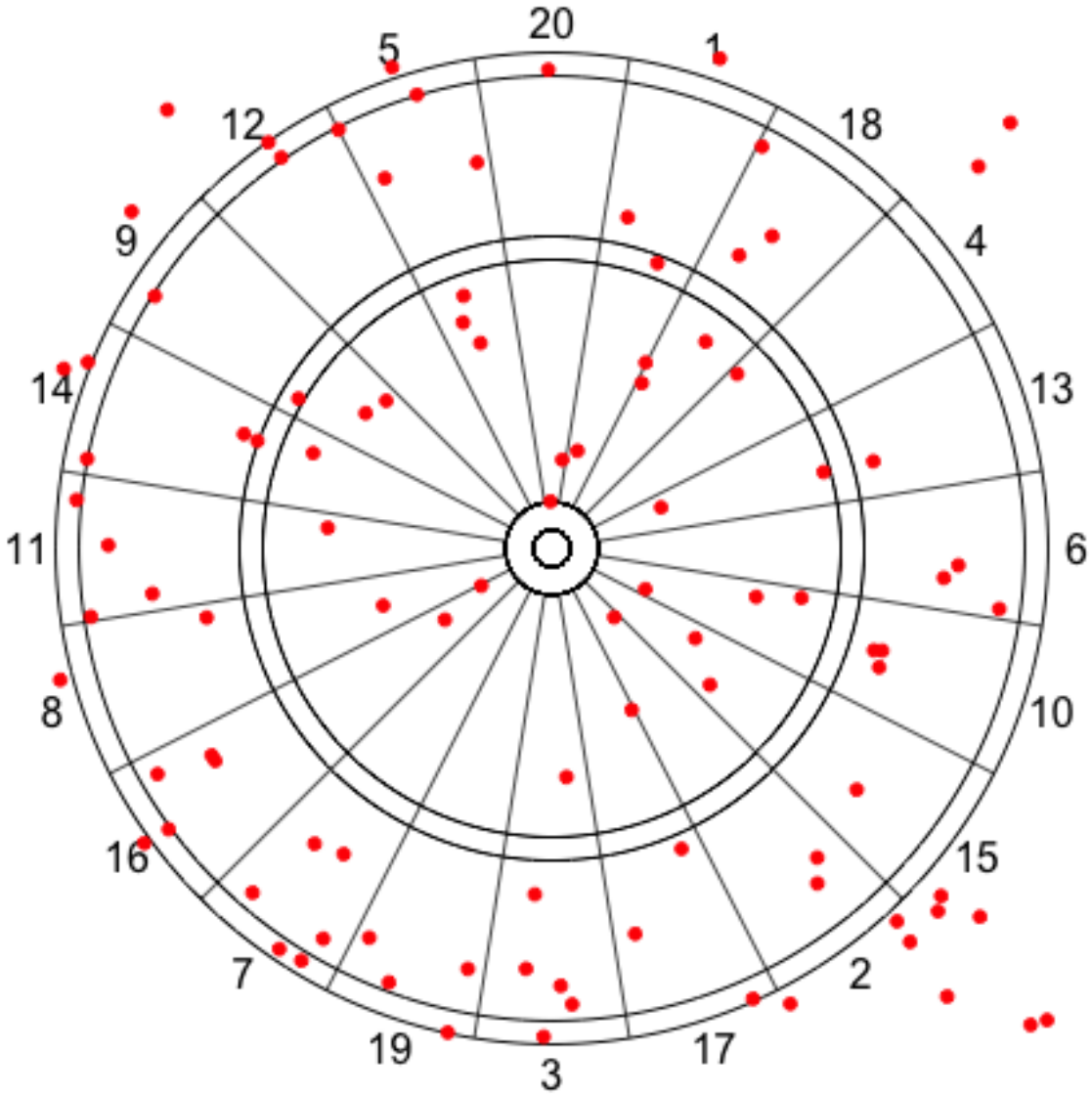
- Generate 100 throws from this model.
- Plot the throw positions on the dart board, with `pch` $= 20$ (to make small solid dots).
- Compute the scores of the 100 throws and add the scores corresponding to each throw on the plot.

Let's Simulate Dart Throws

Solution

```
> # We simulate 100 throws with the x and y
> # location modeled by uniform on (-R, R).
> throws  <- 100
> R <- 170
> x <- runif(throws, min = -R, max = R)
> y <- runif(throws, min = -R, max = R)
> drawBoard(board)
> points(x, y, pch = 20, col = "red")
```

Let's Simulate Dart Throws

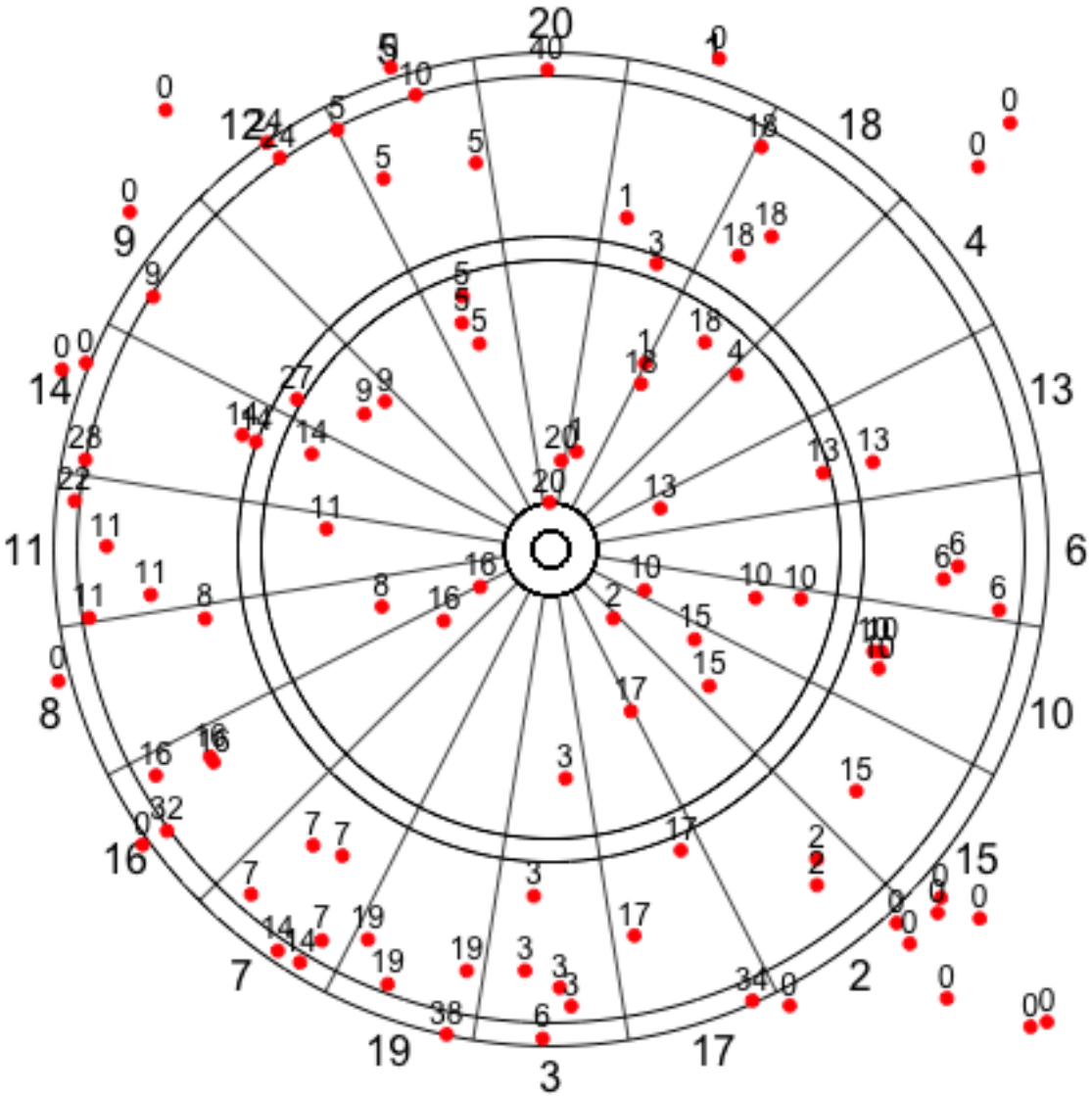


Let's Simulate Dart Throws

Our Solution

```
> # We score our throws and add the values to the plot
> scores <- scorePositions(x, y, board)
> text(x, y + 8, scores, cex = .75)
```

Let's Simulate Dart Throws



Which Strategy is Better?

Our Task

Now let's simulate $10,000$ dart throws according to the two possible models:

1. $\texttt{x}$ and $\texttt{y}$ are both $N(0, 50^2)$.
2. $\texttt{x}$ and $\texttt{y}$ are both $Uniform(-R, R)$.

Which Strategy is Better?

Our Task

Now let's simulate $10,000$ dart throws according to the two possible models:

1. `x` and `y` are both $N(0, 50^2)$.
2. `x` and `y` are both $Uniform(-R, R)$.

```
> throws <- 10000
> x1 <- rnorm(throws, sd = std.dev)
> y1 <- rnorm(throws, sd = std.dev)
> x2 <- runif(throws, min = -R, max = R)
> y2 <- runif(throws, min = -R, max = R)
```

Which Strategy is Better?

Our Task

What are the average scores under each model, and which is higher?

Which Strategy is Better?

Our Task

What are the average scores under each model, and which is higher?

```
> scores1 <- scorePositions(x1, y1, board)
> scores2 <- scorePositions(x2, y2, board)
> mean(scores1)

[1] 12.3174

> mean(scores2)

[1] 10.028
```

Check Yourself

Tasks

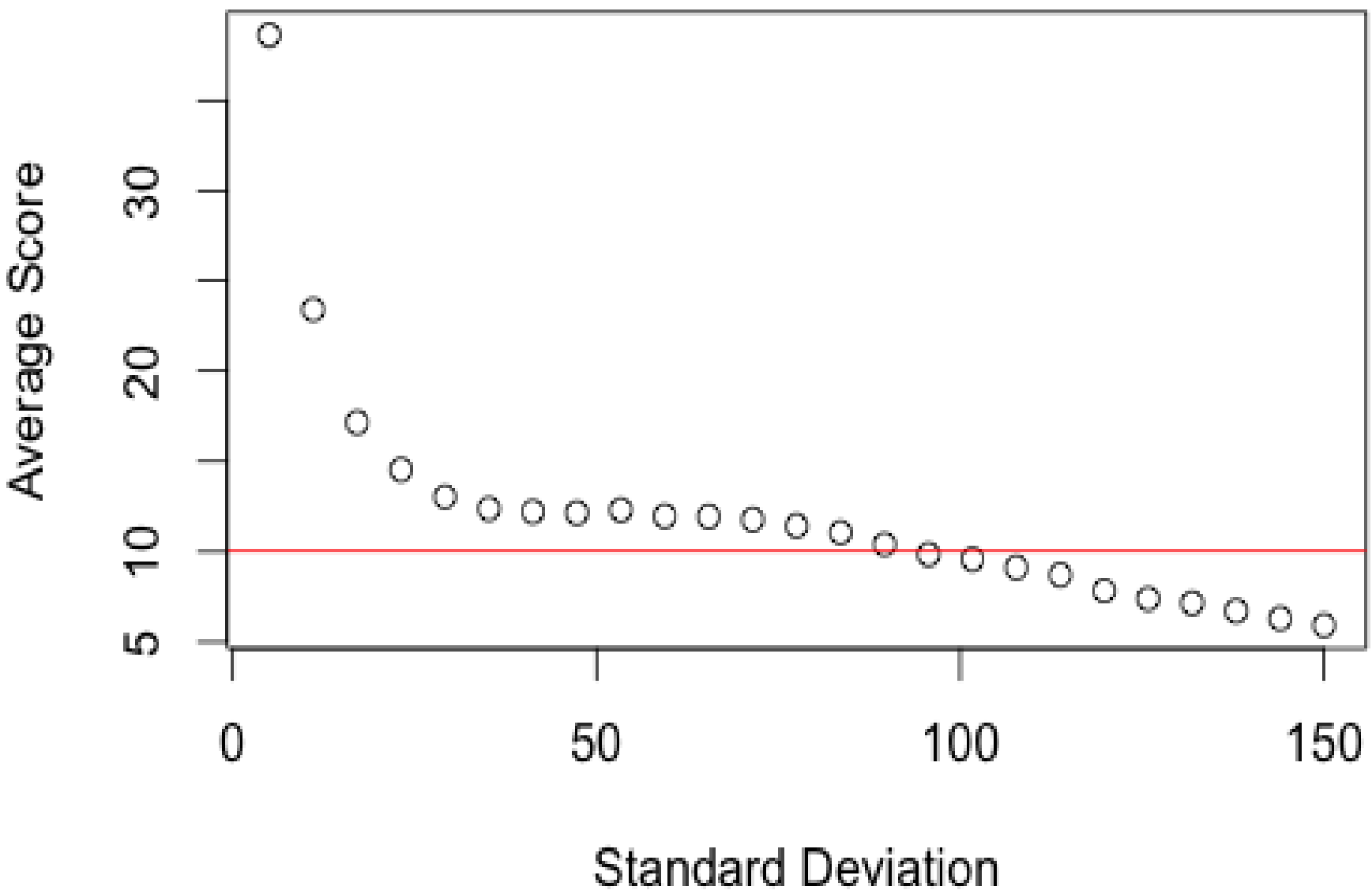
- For 25 values of standard deviation (sd) between 5 mm and 150 mm, draw $10,000$ throws from model (1). For each value of sd, compute the average score.
- Make a plot showing the average score as a function of sd. Label the axes appropriately, and draw the average score calculated for the uniform model (2) as a horizontal line, in red.
- At what value of sd does it become better to throw uniformly at the board?

Check Yourself

Solutions

```
> sd.values  <- seq(5, 150, length.out = 25)
> n          <- length(sd.values)
> avg.scores <- rep(NA, n)
> names(avg.scores) <- round(sd.values,1)
> for (i in 1:n) {
+   x.throws <- rnorm(throws, sd = sd.values[i])
+   y.throws <- rnorm(throws, sd = sd.values[i])
+   scores   <- scorePositions(x.throws, y.throws, board)
+   avg.scores[i] <- mean(scores)
+ }
> plot(sd.values, avg.scores, xlab = "Standard Deviation",
+      ylab = "Average Score")
> abline(mean(scores2), 0, col = "red")
```

Let's Simulate Dart Throws



Check Yourself

Consider fixed the value sd $= 35$ mm, and consider a normal model for throws where $\mathtt{x}$ is $N(mean.x, 35^2)$ and $\mathtt{y}$ is $N(mean.y, 35^2)$, with

1. (mean.x, mean.y) $= (0, 0)$
2. (mean.x, mean.y) $= (0, 103)$
3. (mean.x, mean.y) $= (-32, -98)$

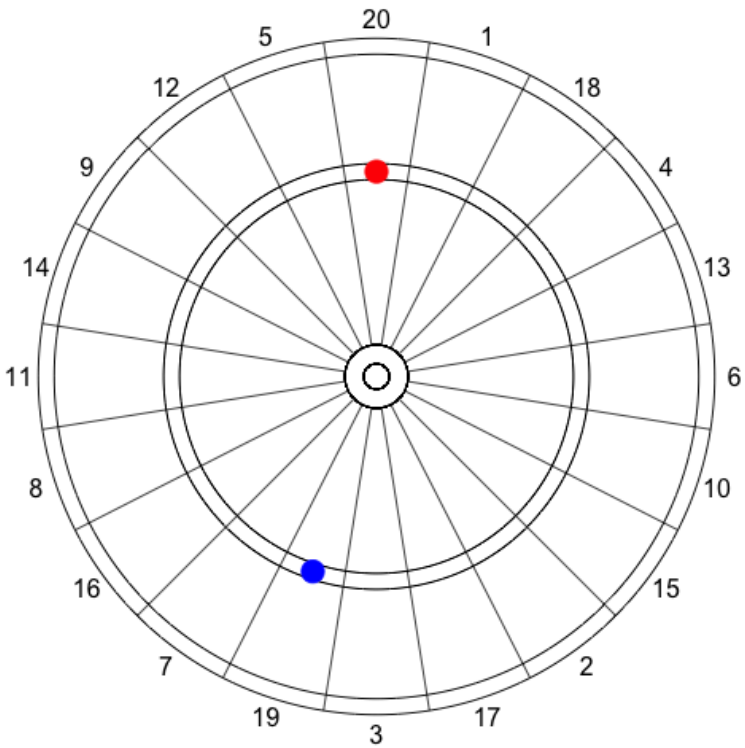
Tasks

- Either by plotting or by looking at scores, determine where the throws are being aimed in each of the three models. That is, the choice of (mean.x, mean.y) $= (0, 0)$ in model (1) means that we are aiming at the center of the board. Where are we aiming in models (2) and (3)?
- For each of the models, draw $10,000$ throws, and compute the average score. How do they compare? Can you explain the results from an intuitive perspective?

Check Yourself

Solution

```
> drawBoard(board)
> points(c(0, -32), c(103, -98), col = c("red", "blue"),
+        pch = 20, cex = 3)
```



Check Yourself

Solution

```
> normal.score <- function(mean.x, mean.y, sd, board) {
+   x1 <- rnorm(throws, mean = mean.x, sd = sd)
+   y1 <- rnorm(throws, mean = mean.y, sd = sd)
+   return(mean(scorePositions(x1, y1, board)))
+ }
> std.dev <- 35
> normal.score(0, 0, std.dev, board)

[1] 12.3721

> normal.score(0, 103, std.dev, board)

[1] 12.8175

> normal.score(-32, -98, std.dev, board)

[1] 13.7954
```

Rejection Sampling

- Normal model is too pessimistic compared to uniform model: normal model can produce throws far from the center, but uniform model is restricted to [-R,R] in each direction.
- Fix by restricting normal model so each sampled coordinate lies in [-R,R].
- Use **rejection sampling** to do this: sample from a normal distribution, but if the draw lies outside of [-R,R], then toss it out.

Rejection Sampling

Our Tasks

Write a function `rnorm.reject()` that takes in the arguments:

- `n`, `mean`, `sd`: just as `rnorm()` does
- `min.val`, `max.val`: upper and lower limits for rejection sampling.

The function should return a sample of length `n` using the rejection sampling method.

Rejection Sampling

Our Solution

```
> rnorm.reject <- function(n, mean = 0, sd = 1,
+                          min.val, max.val) {
+   all.samps <- c()
+   while (length(all.samps) < n) {
+     samp <- rnorm(1, mean = mean, sd = sd)
+     if (min.val < samp & samp < max.val) {
+       all.samps <- c(all.samps, samp)
+     }
+   }
+   return(all.samps)
+ }
```

Rejection Sampling

Our Solution

```
> sampled.vals <- rnorm.reject(n = throws, sd = std.dev,
+                              min.val = -R, max.val = R)
> summary(sampled.vals)

     Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
-121.9000  -24.0900   -0.3615   -0.5439   23.0100  129.2000
```

Check Yourself

Tasks

- For 25 values of standard deviation (sd) between 5 mm and 150 mm, draw 10,000 throws from model (1), i.e. `x` and `y` are both $N(0, 50^2)$, but use `rnorm.reject()`. For each value of sd, compute the average score.
- Make a plot showing the average score as a function of sd. Label the axes appropriately, and draw the average score calculated for the uniform model (2) as a horizontal line, in red.
- At what value of sd does it become better to throw uniformly at the board?

Check Yourself

Solutions

```
> sd.values  <- seq(5, 150, length.out = 25)
> n          <- length(sd.values)
> avg.scores <- rep(NA, n)
> names(avg.scores) <- round(sd.values,1)
> for (i in 1:n) {
+   x.throws <- rnorm.reject(throws, sd = sd.values[i],
+                            min.val = -R, max.val = R)
+   y.throws <- rnorm.reject(throws, sd = sd.values[i],
+                            min.val = -R, max.val = R)
+   scores   <- scorePositions(x.throws, y.throws, board)
+   avg.scores[i] <- mean(scores)
+ }
> plot(sd.values, avg.scores, xlab = "Standard Deviation",
+      ylab = "Average Score")
> abline(mean(scores2), 0, col = "red")
```

Let's Simulate Dart Throws