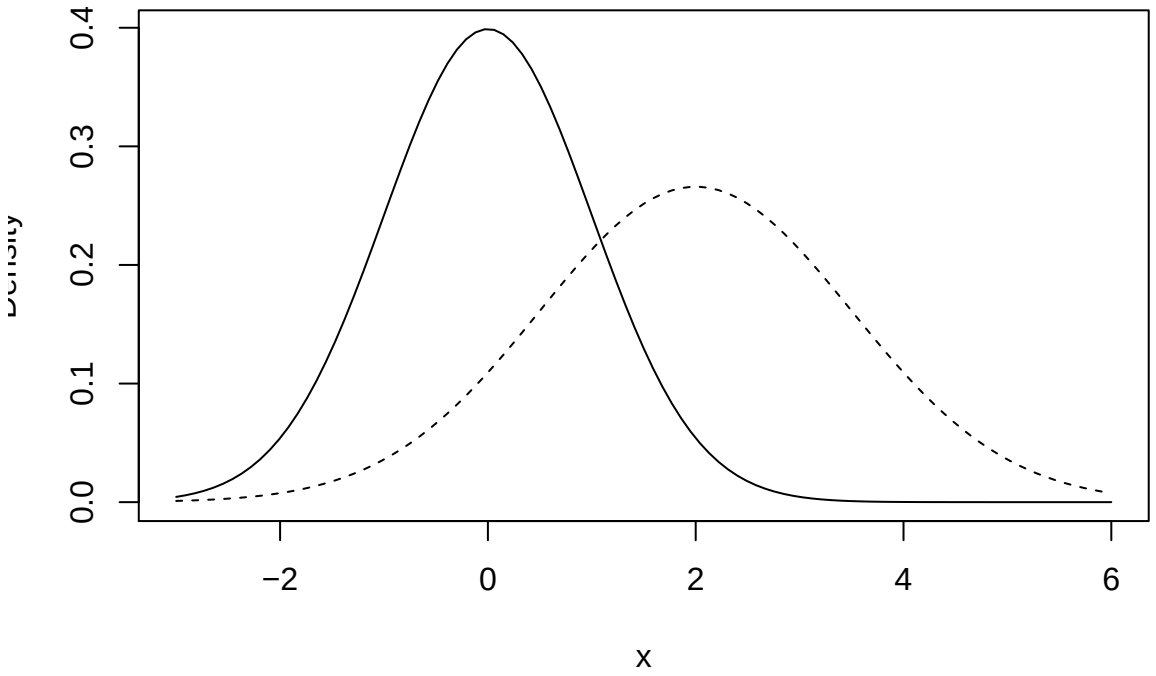
HW1-33B

Contents

0.1 Call Expressions

The standard normal curve is centered at 0 and has a spread, AKA standard deviation, of 1. This curve is drawn in the figure below along with a normal curve that has a center of 2 and a spread of 1.5 (dashed curve).

```
curve(dnorm, from = -3, to = 6, ylab = "Density")
curve(dnorm(x, mean = 2, sd = 1.5), add = TRUE, lty = 2)
```



The family of normal curves all have the same shape and each is uniquely specified by its center (mean) and spread (standard deviation)

1. Read the documentation for `rnorm()` to figure out how to call `rnorm()` to generate 5 random values from a normal distribution with mean 17 and standard deviation 3.

```
res = rnorm(5, mean=17, sd=3)
```

Debug: Result is 20.1647812, 15.7668418, 22.0226578, 12.4548087, 20.4037194.

2. Create a vector called `normSamps` containing 1000 random values from a normal distribution with mean 1 and standard deviation 2.

```
set.seed(66)
normSamps = rnorm(1000, mean=1, sd=2)
```

3. The 10% trimmed mean is obtained by taking the mean of the middle 80% of the values (removing the lowest 10% and the highest 10% of the values). Find the 10% trimmed mean of normSamps and assign this value to `mean10`. Hint: Look at the help file for the mean() function.

```
mean10 = mean(normSamps, trim=0.1)
```

Debug: Result is 1.0470772.

4. Use `sum()` to calculate the fraction of values in `normSamps` that are less than 3. Write your solution so that it coerces a logical vector to numeric without using the function as.numeric(). Write your code generically so that it doesn't depend on `normSamps` being a vector of 1000 elements. Assign the fraction to the variable `frac3` Hint: consider using the `length()` function.

```
# It maybe slow and expensive to create a useless vector by filter.
#   (REALLY REALLY slow in traditional programming language, but maybe there's strange optimization in
isLessThan3 = Vectorize(function(ele) {
    return(ele < 3) # Strange R syntax...
})
frac3 = length(normSamps[isLessThan3(normSamps)]) / length(normSamps)
```

Debug: Result is 0.839.

5. A percentile is the value at which a given percentage of samples fall at or below it and is sometimes synonomously called a quantile, e.g., the 90th percentile is the value at which 90% of data values fall at or below it. Calculate the 90th percentile of `normSamps` and assign this value to `per90`. Hint: The `quantile()` function is helpful for this question.

```
per90 = quantile(normSamps, probs=c(0.9))
```

Debug: Result is 3.5420231.

6. Calculate the greatest absolute distance from 1 for the values in `normSamps`. Assign this value to a variable called `maxDev`, short for "maximum deviation" from the mean. Hint: The functions abs() and max() are helpful for this question.

```
maxDev = max(abs(normSamps - 1))
```

Debug: Result is 7.0159969.

0.2 Working with Vectors

Run the following code chunk to load the vectors for our 14-member family:

```
load(url("http://www.stat.berkeley.edu/users/nolan/data/afamily.rda"))
```

7. Calculate `bmi` from `fheight` and `fweight` by dividing weight in pounds by height in inches squared and multiplying by a conversion factor of 703. Assign this to `bmi_alt`

```
bmi_alt = 703 * fweight / (fheight^2)
```

Debug: Result is 25.1071429, 21.2822266, 24.4051417, 24.4303854, 18.5149153, 28.8862457, 28.1260813, 20.632426, 26.6057526, 29.9831383, 25.9964357, 22.594123, 24.207989, 22.8603018.

8. Check that `fbmi` and `bmi_alt` match using the functions `==` and `all()`

```
isMatch = all(fbmi == bmi_alt)
```

Debug: Result is FALSE, because fbmi is 25.162388, 21.3290554, 24.4588421, 24.4841414, 18.5556551, 28.9498062, 28.1879692, 20.6778251, 26.6642952, 30.0491124, 26.0536376, 22.6438386, 24.2612556, 22.910603.

9. Suppose that as long as the entries in `bmi_alt` and `fbmi` are within 0.1 of each other, then we consider these two to be equivalent. Calculate the differences and assign it to `bmi_diff`.
   Check whether all of the differences are within 0.1 of each other. Produce a logical vector of length 1 called `bmi_same` that is TRUE if the entries in `bmi_alt` and `fbmi` are within 0.1 of each other.

```
bmi_diff = abs(fbmi - bmi_alt)
bmi_same = all(bmi_diff < 0.1)
```

Debug: bmi_diff is 0.0552451, 0.0468289, 0.0537005, 0.053756, 0.0407398, 0.0635606, 0.0618879, 0.0453991, 0.0585426, 0.0659741, 0.0572019, 0.0497155, 0.0532667, 0.0503012, and bmi_same is TRUE.

10. Create a vector of last names for the family members, called `flast`. The values should be `Smith` for each entry. Hint: use `rep()` and `length()` to do this.

```
flast = rep('Smith', length(fbmi))
```

Debug: flast is Smith, Smith, Smith, Smith, Smith, Smith, Smith, Smith, Smith, Smith, Smith, Smith, Smith, Smith.