MTH 2520 R Notes 10

1 Doing Statistics, Math, and Simulations in R

1.1 Correlation

1.1.1 Definition and Interpretation of Correlation

- The ***correlation*** between two variables $x_1, x_2, \ldots, x_n$ and $y_1, y_2, \ldots, y_n$ is a statistic that measures the **strength** and **direction** of the **association** between them.

- It's denoted by $\boldsymbol{r}$ and defined as

$$r \;=\; \frac{1}{n-1}\left(\frac{\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y})}{s_x s_y}\right)$$

  where $\bar{x}$, $\bar{y}$, s_x, and s_y are the means and standard deviations of the x_i and y_i values, respectively.

- r has the following **properties and interpretations**:

  1. $-1 \leq r \leq 1$ no matter what the x_i and y_i values are.
  2. The **sign** of r tells us the **direction** of the linear association between x and y:
     - ◇ $r > 0$ indicates a **positive association** between x and y.
     - ◇ $r < 0$ values indicate a **negative association**.
  3. The **value** of r tells us the **strength** of the linear association between x and y is:
     - ◇ $r \approx 0$ implies a **weak association** or none at all.
     - ◇ $r \approx 1.0$ or $r \approx -1.0$ implies a **strong association**.
     - ◇ $r = 1.0$ and $r = -1.0$ only when there's a **perfect linear association**.

1.1.2 Computing the Correlation in R

- To compute the correlation in R we use:

```
cor()            # Compute the correlation between x and y
```

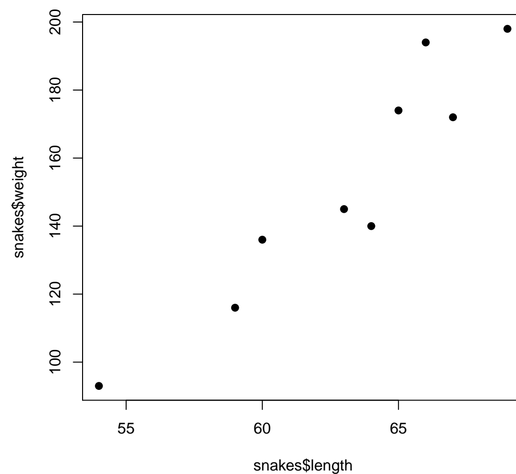
- `cor()` takes two vector arguments `x` and `y` and returns the correlation r. For example, here's a data frame containing lengths (cm) and weights (grams) of $n = 9$ female snakes.

```
> snakes
```

```
  snake length weight
1     1     60    136
2     2     69    198
3     3     66    194
4     4     64    140
5     5     54     93
6     6     67    172
7     7     59    116
8     8     65    174
9     9     63    145
```

Here's a **scatterplot** of the lengths and weights (using the `plot()` function):

```
> plot(x = snakes$length, y = snakes$weight, pch = 19)
```



The correlation between the length and weight is:

```
> cor(x = snakes$length, y = snakes$weight)
```

```
[1] 0.9436756
```

reflecting the strong, positive association seen in the plot.

1.1.3 Correlation Matrices

- If we pass a *data frame* (or *matrix*) to `cor()`, it computes correlations for every pair of variables, and returns the results in a ***correlation matrix***. For example, here's a data frame `my.data`:

  ```
  > my.data
  ```

```
    Y X1  X2  X3
1  24 10 2.4 100
2  22 11 1.9  75
3  25 13 2.3  68
4  22 12 1.8  77
5  26 13 1.3  80
6  26 16 0.9  81
7  25 15 1.2  72
8  27 20 1.0  55
9  30 18 1.3  39
10 33 19 1.0  67
11 29 19 0.8  77
12 30 22 0.7  94
```

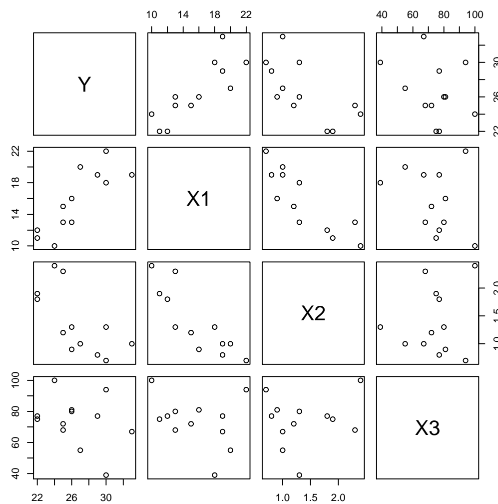
and here's the ***correlation matrix***:

```
> cor(my.data)
```

```
            Y         X1         X2         X3
Y   1.0000000  0.8290126 -0.6693360 -0.2968452
X1  0.8290126  1.0000000 -0.8556911 -0.3026625
X2 -0.6693360 -0.8556911  1.0000000  0.1620926
X3 -0.2968452 -0.3026625  0.1620926  1.0000000
```

- Here's the corresponding ***scatterplot matrix*** made using the function `pairs()`:

```
> pairs(my.data)
```



Section 1.1 Exercises

Exercise 1 The vectors `illit` and `murder` contain illiteracy and murder rates for each of the 50 states in the U.S.:

```
> illit <- state.x77[ , 3]
> murder <- state.x77[ , 5]
```

a) Use `plot()` to make a scatterplot of the murder rates (y-axis) versus illiteracy rates (x-axis).

b) Guess the correlation between murder and illiteracy rates from the scatterplot, then use `cor()` to compute it.

Exercise 2 The correlation r only measures the strength of a *linear* relationship between x and y. In particular, x and y can have a strong *curved* relationship but r close to zero. Consider the following variables:

```
> x <- c(7.1, 2.5, 3.9, 0.9, 9.6, 0.1, 5.7, 7.6, 8.7, 0.4)
> y <- c(4.4, 6.2, 1.2, 16.8, 21.2, 24.0, 0.5, 6.8, 13.7, 21.2)
```

a) Use `plot()` to make a scatterplot of the data.

b) Use `cor()` to compute the correlation.

c) Why is the correlation so close to zero despite the strong relationship between x and y?

Exercise 3 The correlation r is affected by the presence of ***outliers*** (observations that lie outside the overall pattern of the data). Consider the following variables:

```
> x <- c(7.1, 2.5, 3.9, 0.9, 9.6, 0.1, 5.7, 7.6, 8.7, 0.4)
> y <- c(1.6, 2.3, 2.4, 2.1, 3.0, 2.0, 2.5, 2.8, 2.9, 2.0)
```

a) Use `plot()` to make a scatterplot of the data. Note the outlier at the bottom.

b) Use `cor()` to compute the correlation between x and y.

c) How would the value of r change if the outlier was removed from the data? Try computing the r with the outlier omitted:

```
> cor(x[-1], y[-1])
```

Exercise 4 Here's a data frame:

```
> y <- c(0.25, 0.24, 0.23, 0.21, 0.21, 0.22, 0.25, 0.26, 0.27, 0.29, 0.29,
         0.28, 0.27, 0.23, 0.24, 0.23, 0.27, 0.27, 0.27, 0.23, 0.23, 0.25,
         0.26, 0.26, 0.26, 0.26, 0.27)
> x1 <- c(0.96, 0.93, 0.84, 0.67, 0.44, 0.22, 0.11, 0.11, 0.24, 0.45, 0.71,
         0.85, 0.95, 0.66, 0.64, 0.65, 0.67, 0.64, 0.64, 0.40, 0.36, 0.35,
         0.34, 0.34, 0.35, 0.38, 0.42)
> x2 <- c(0.48, 0.33, 0.22, 0.13, 0.12, 0.20, 0.40, 0.63, 0.81, 0.89, 0.90,
         0.79, 0.65, 0.36, 0.35, 0.38, 0.65, 0.64, 0.67, 0.33, 0.37, 0.42,
         0.46, 0.53, 0.60, 0.65, 0.69)
> my.data <- data.frame(y, x1, x2)
```

a) After creating the data frame `my.data`, use `cor()` to compute the *correlation matrix* of the variables in the data set.

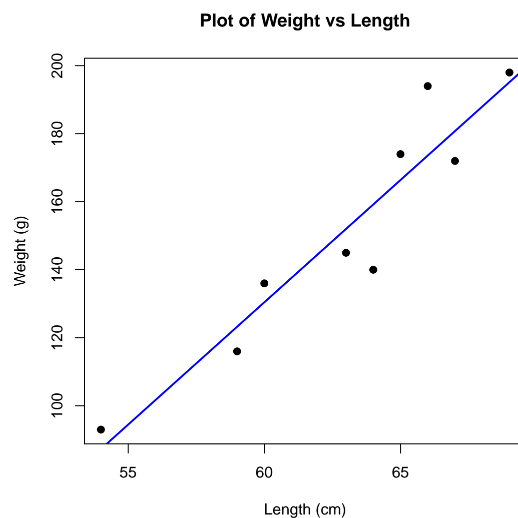
b) Among the variables `y`, `x1`, and `x2`, which pair has the highest correlation? Which pair has the lowest?

c) Use `pairs()` to create a *scatterplot matrix* of the data. Do you notice any "interesting" patterns?

1.2 Linear Regression

1.2.1 The Method of Least Squares

- A ***linear regression analysis*** refers to fitting a straight line to a scatterplot, as shown below.



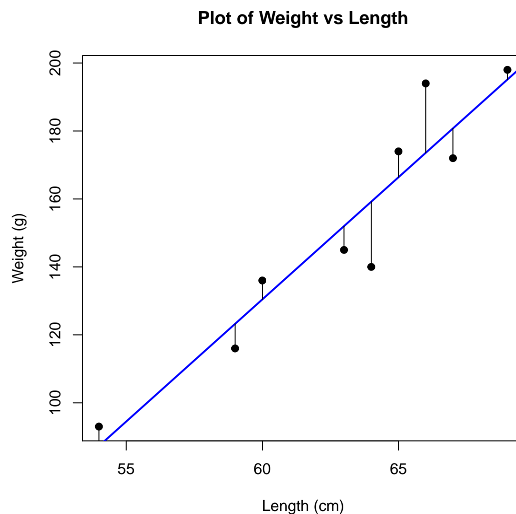
- The ***fitted regression line***, which we write as

$$\hat{y} = b_0 + b_1 x, \tag{1}$$

is obtained using the ***method of least squares***, whereby the slope b_1 and y-intercept b_0 are selected so as to minimize the ***sum of squared errors*** (deviations away from the line),

$$\sum_{i=1}^{n}(y_i - (b_0 + b_1 x_i))^2$$

(The errors $y_i - (b_0 + b_1 x_i)$ are the vertical lines in the plot below.)



- Once the regression line has been fitted to the data, the errors are called ***residuals*** and denoted $\boldsymbol{e_i}$, i.e.

$$e_i \;=\; i\text{th residual} \;=\; y_i - (b_0 + b_1 x_i).$$

A residual will be positive if the point lies above the line, and negative if it lies below the line.

1.2.2 Carrying Out a Regression Analysis in R

- To obtain the regression line in R we use:

```
lm()        # Fit a linear regression model to a set of data (and carry
            # out t tests for the slope and intercept)
summary()   # Look at a summary of the regression analysis
```

- The ”linear model” function `lm()` takes a ***formula*** as it’s main argument and an optional argument `data` (a data frame containing the variables used in the formula). Here’s an example using the `snakes` data:

```
> my.reg <- lm(weight ~ length, data = snakes)
```

Above, the *formula* `weight ~ length` indicates that `weight` is the ***response variable*** (y) and `length` is the ***predictor*** (x).

- To look at the regression analysis results, we type:

```
> summary(my.reg)

Call:
lm(formula = weight ~ length, data = snakes)

Residuals:
    Min      1Q  Median      3Q     Max
-19.192  -7.233   2.849   5.727  20.424

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -301.0872    60.1885  -5.002 0.001561 **
length         7.1919     0.9531   7.546 0.000132 ***
---
Signif. codes:  0 Ś***Š 0.001 Ś**Š 0.01 Ś*Š 0.05 Ś.Š 0.1 Ś Š 1

Residual standard error: 12.5 on 7 degrees of freedom
Multiple R-squared:  0.8905,       Adjusted R-squared:  0.8749
F-statistic: 56.94 on 1 and 7 DF,  p-value: 0.0001321
```

- We'll only be concerned with the following items from the output above:
  - `Residuals:` A summary of the residuals.
  - `Coefficients:` The intercept and slope of the fitted regression line, $b_0 = -301.0872$ and $b_1 = 7.1919$. Thus the equation of line is

$$\hat{y} \;=\; -301.0872 + 7.1919\, x.$$

    (The other information, `Std. Error`, `t value`, and `Pr(>|t|)`, are related to hypothesis tests for the coefficients).
  - `Residual standard error:`

$$\text{Residual standard error} \;=\; \sqrt{\frac{1}{n-2}\sum_{i=1}^{n} e_i^2}$$

    This represents the size of a typical residual, in this case 12.5 grams.
  - `Multiple R-squared:` The so-called ***coefficient of determination*** (usually denoted $\boldsymbol{R^2}$), a measure of how well the line fits the data:
    * $R^2 \approx 1$ indicates that the line fits the data well and that most of the variation in y is explained by x.

  * $R^2 \approx 0$ indicates that the line doesn't fit the data very well and that very little of the variation in y is explained by x.

  Above, $R^2 = 0.8905$ indicates that the line fits the date well and that most of the variation in snakes' weights (89.05% in fact) is explained by variation in their lengths.

- The object `my.reg` returned by `lm()` is a list:

```
> is.list(my.reg)

[1] TRUE

> names(my.reg)

 [1] "coefficients"  "residuals"     "effects"       "rank"          "fitted.values"
 [6] "assign"        "qr"            "df.residual"   "xlevels"       "call"
[11] "terms"         "model"
```

- To extract specific elements, use the `$` operator. For example:

```
> my.reg$coefficients

(Intercept)      length
 -301.08721     7.19186

> my.reg$residuals

         1          2          3          4          5          6          7          8
  5.575581   2.848837  20.424419 -19.191860   5.726744  -8.767442  -7.232558   7.616279
         9
 -7.000000
```

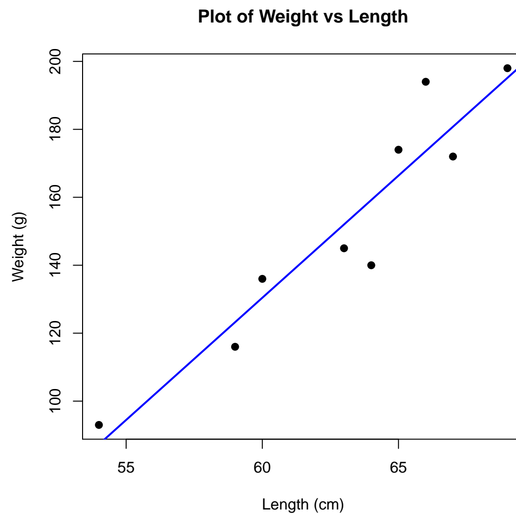
1.2.3 Adding the Regression Line to a Scatterplot

- To add the regression line to a scatterplot of the data, we first create the plot using `plot()`:

```
> plot(x = length, y = weight, pch = 19, xlab = "Length (cm)",
       ylab = "Weight (g)", main = "Plot of Weight vs Length")
```

  then add the line using `abline()`:

```
> abline(my.reg, col = "blue", lwd = 2)
```



Section 1.2 Exercises

Exercise 5 The equation of the regression line fitted to the `snakes` data is:

$$\hat{y} = -301.0872 + 7.1919\,x.$$

a) By plugging an x value into the equation, we're able to *predict* the value of y. What would you predict for the weight of snake that's 62 cm long?

b) The *slope* of the fitted line indicates a *typical change* in y for a given change in x. What's a typical weight gain when a snake grows 1 cm longer?

Exercise 6 The vectors `illit` and `murder` contain illiteracy and murder rates for each of the 50 states in the U.S.:

```
> illit <- state.x77[ , 3]
> murder <- state.x77[ , 5]
```

a) Use `lm()` to carry out a regression analysis, with y = murder rate and x = illiteracy rate. Write the R command(s) you used.

b) Use `summary()` to look at the results of the regression analysis. Give the equation of the fitted regression line.

c) Use `plot()` to make a scatterplot of the murder rates (y-axis) versus illiteracy rates (x-axis). Then use `abline()` to add the regression line to the scatterplot. Write out the R command(s) you used.

d) By plugging an x value into the equation of the line, we're able to ***predict*** the value of y. What would you predict the murder rate to be for a state whose illiteracy rate is 2.5?

e) The *slope* of the fitted line indicates a *typical change* in y for a one-unit change in x. How much does the murder rate typically increase one-percentage point increase in the illiteracy rate?

f) The *residuals* are the vertical deviations of the points in the scatterplot away from the fitted line. The `Residual standard error` represents the size of a typical residual. What's the size of a typical residual for the murder and illiteracy rates data?

1.3 Random Numbers and Simulation

1.3.1 Sampling Elements from a Vector

- To generate a random sample from the elements of a vector, we use:

```
sample()      # Generate a random sample from the elements of a vector.
```

- `sample()` takes arguments `x`, a vector, and `size`, the sample size. It returns a sample drawn ***without replacement*** from the elements of `x`. An optional argument `replace` can be set to `TRUE` to sample ***with replacement***.

- For example, to generate a random sample of size $n = 10$ from the numbers $1, 2, \ldots, 100$, type:

```
> sample(x = 1:100, size = 10)

 [1] 19 25 58 34 23 63 84 56 81 87
```

- Setting `size` equal to `length(x)` returns a random permutation (ordering) of the elements of `x`. For example, to place the numbers $1, 2, 3, 4, 5$ in a random order, type:

```
> sample(x = 1:5, size = 5)

[1] 2 5 1 4 3
```

1.3.2 Duplicating a Random Sample Using `set.seed()`

- This function enables us to *reproduce* a set of random numbers later:

```
set.seed()      # Set the random seed, which determines which numbers
                # will be generated by R's random number generator.
```

- `set.seed()` takes a positive integer argument `seed` (any value will do) that determines which numbers will be generated by R's random number generator.

- For example, below we set the "seed" to 15 *before* using `sample()`:

```
> set.seed(15)
> sample(x = 1:100, size = 10)

 [1] 61 20 95 64 36 94 77 24 97 76
```

  Now to regenerate the *same* set of random numbers, we set the "seed" to 15 again before generating the sample:

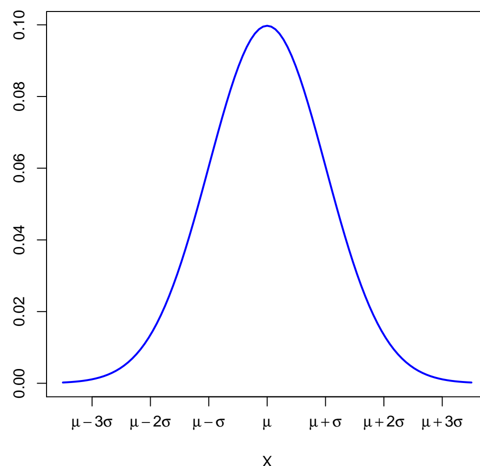
```
> set.seed(15)
> sample(x = 1:100, size = 10)

 [1] 61 20 95 64 36 94 77 24 97 76
```

1.3.3 Generating Uniform and Normal Random Numbers

- A ***uniform(a, b) random variable*** is a randomly generated number that's equally likely to fall anywhere on the interval (a, b).

- A ***normal(μ, σ) random variable*** is a number generated randomly according to probabilities given by the bell-shaped curve with mean μ and standard deviation σ:



  The normal curve above has $\mu = 50$ and $\sigma = 4$. In general:

  - μ determines where the curve is centered.
  - σ determines how spread out the curve is: The curve extends about three σs to the left of μ and about three σs to the right of μ (total spread = six σs).

- To generate *uniform* or *normal* random variables, use:

```
runif()        # Generate a random sample of size n from a uniform
               # probability distribution.
rnorm()        # Generate a random sample of size n from a normal
               # probability distribution.
```

- `runuf()` takes arguments `n`, `min`, and `max`, and returns `n` numbers randomly selected on the interval from `min` to `max`.
- For example, to generate $n = 3$ uniform(0, 1) values, type:

```
> runif(n = 3, min = 0, max = 1)

[1] 0.4161184 0.6947637 0.1488006
```

- `rnorm()` takes arguments `n`, `mean`, and `sd`, and returns `n` numbers randomly generated from the normal(μ, σ) curve, with μ and σ specified via `mean` and `sd`.
- For example, to generate $n = 3$ values from the normal(μ, σ) curve, with $\mu = 4$ and $\sigma = 2$, we type:

```
> rnorm(n = 3, mean = 4, sd = 2)

[1] 7.814325 6.289754 2.470939
```

1.3.4 Computing Uniform and Normal Probabilities

- To compute the (cumulative) probability that a uniform or normal random variable will be *less* than a value `q`, use:

```
punif()       # Compute the probability that a uniform random variable
              # will fall below some value q.
pnorm()       # Compute the probability that a normal random variable
              # will fall below some value q.
```

- `punif()` takes arguments `q`, `min`, and `max`, and returns the probability a value randomly selected from the interval from `min` to `max` will fall below `q`.
- For example, the probability that a randomly selected value between 0 and 10 will fall below 4.5 is:

```
> punif(q = 4.5, min = 0, max = 10)

[1] 0.45
```

Thus the probability is 0.45, or 45%.

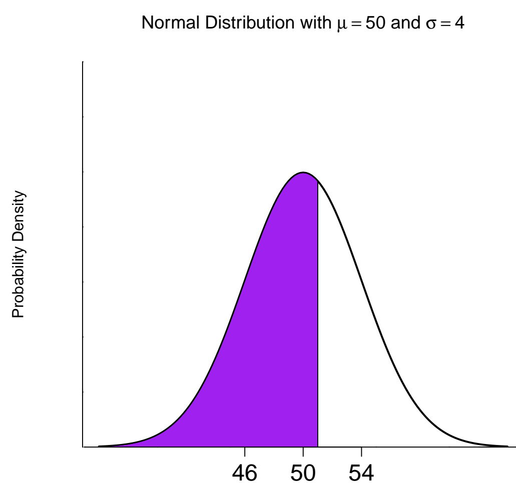
- `pnorm()` takes arguments `q`, `mean`, and `sd`, and returns the probability that a value randomly selected from the normal(μ, σ) curve, with μ and σ specified via `mean` and `sd`, will fall below `q`.

- For example, to compute the probability that a normal(μ, σ) random variable, with $\mu = 50$ and $\sigma = 4$, will be less than 51, we type:

```
> pnorm(q = 51, mean = 50, sd = 4)
```

```
[1] 0.5987063
```

  Thus the probability is 0.59871, or about 59.9%. This is the shaded area in the figure below.



- Since the probability of something *not* happening is *one minus* the probability of it happening, for *upper* probabilities, we subtract the *lower* probability from 1.

  For example, the probability that the value will be *greater* than 51 is *one minus* the probability that it will be *less* that 51, obtained by typing:

```
> 1 - pnorm(q = 51.0, mean = 50, sd = 4)
```

```
[1] 0.4012937
```

1.3.5 Computing Uniform and Normal Percentiles (Quantiles)

- The **100*pth* *percentile*** (or ***pth* *quantile***) is a value q, below which a random variable will fall with probability p, that is, with a $100p\%$ chance (where $0 < p < 1$).

- To obtain percentiles for uniform and normal random variables, we use:

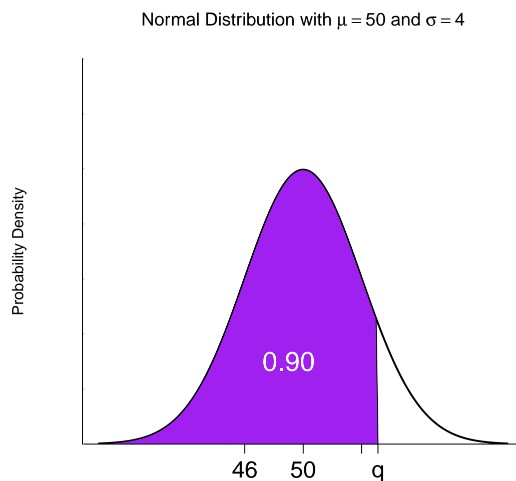
```
qunif()        # Uniform distribution percentiles (quantiles), with
               # arguments p, min, and max
qnorm()        # Normal distribution percentiles (quantiles), with
               # arguments p, mean, and sd
```

- In each case, the argument `p` is a value between 0 and 1 specifying the probability for the percentile we desire.
- For example, to find the **90th percentile** of the normal(μ, σ) curve, with $\mu = 50$ and $\sigma = 4$, we type:

```
> qnorm(p = 0.90, mean = 50, sd = 4)
```

```
[1] 55.12621
```

Thus the percentile is $q = 55.12621$. This is represented by the value q in the figure below.



1.3.6 Obtaining the Density Curve for Uniform and Normal

- To obtain values of the normal(μ, σ) curve (or the uniform "curve") we use:

```
dunif()      # Uniform distribution density function, with arguments x,
             # min, and max
dnorm()      # Normal distribution density function, with arguments x,
             # mean, and sd
```

- The normal(μ, σ) curve is given by the function:

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{-(x-\mu)^2}{2\sigma^2}},$$

and we can obtain $f(x)$ at a value x using `dnorm()`.

- As an example, the height of the normal(μ, σ) curve, with with $\mu = 50$ and $\sigma = 4$, at $x = 46$ is:

```
> dnorm(x = 46, mean = 50, sd = 4)

[1] 0.06049268
```

Thus $f(46) = 0.06049268$.

Section 1.3 Exercises

Exercise 7 Recall that `set.seed()` is used when we want to reproduce a set of random values later.

a) Use `set.seed()` to set the "seed" to a value of your choosing (any positive integer will do, and it doesn't matter which one you use). Then use `sample()` to generate $n = 5$ random numbers from $1, 2, \ldots, 100$.

b) Now set the "seed" again (to the same value), and use `sample()` to regenerate the five random numbers. Confirm that you get the same five values that you got in part *a*.

c) What would've happened in part *b* if you hadn't set the "seed" prior to generating the random values? Try it.

Exercise 8 This problem concerns *uniform* random variables.

a) Use `runif()` to generate $n = 300$ random values between 0 and 1. Save them in a vector `x`.

b) Round the values to one decimal place:

```
> x <- round(x, digits = 1)
```

c) The function `stripchart()` will produce a ***dot plot*** of a data set:

```
> stripchart(x, method = "stack", at = 0)
```

Make a dot plot of your sample of rounded uniform random values from part *b*. Are the generated values fairly evenly spread over the interval from 0 to 1?

Exercise 9 This problem concerns *normal* random variables.

a) Use `rnorm()` to generate $n = 300$ random values from the normal(μ, σ) curve, with with $\mu = 0$ and $\sigma = 1$. Save them in a vector `x`.

b) Round the values to one decimal place:

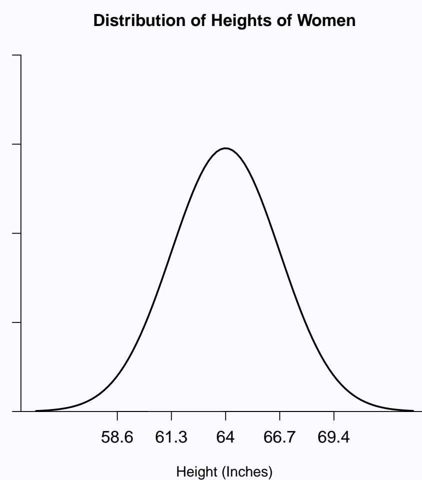
```
> x <- round(x, digits = 1)
```

c) The function `stripchart()` will produce a ***dot plot*** of a data set:

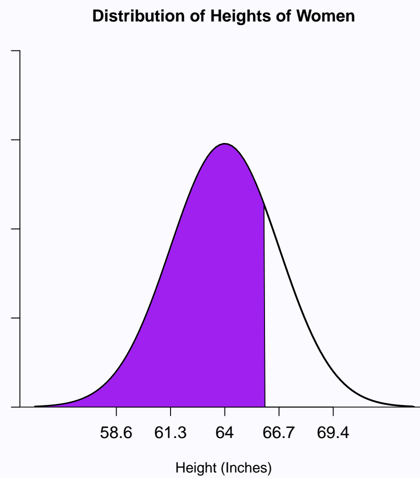
```
> stripchart(x, method = "stack", at = 0)
```

Make a dot plot of your sample of rounded normal random values from part *b*. Do the generated values follow a bell-shaped pattern?

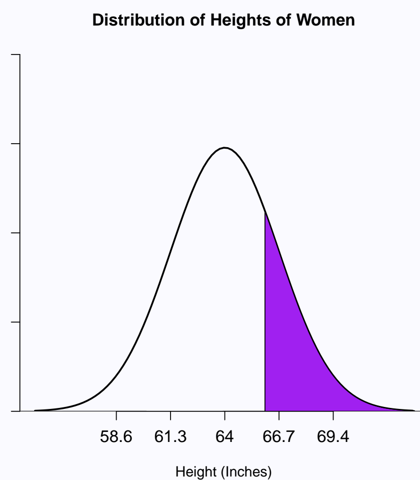
Exercise 10 Heights of women aged 20 to 29 follow a normal(μ, σ) curve with mean μ = 64 inches and standard deviation σ = 2.7 inches, shown below.



a) What percentage of women are shorter than 66 inches (i.e. what's the probability that a randomly selected woman's height will be *less than* than 66 inches)? Use `pnorm()` to find the answer. This is represented by the shaded area below.

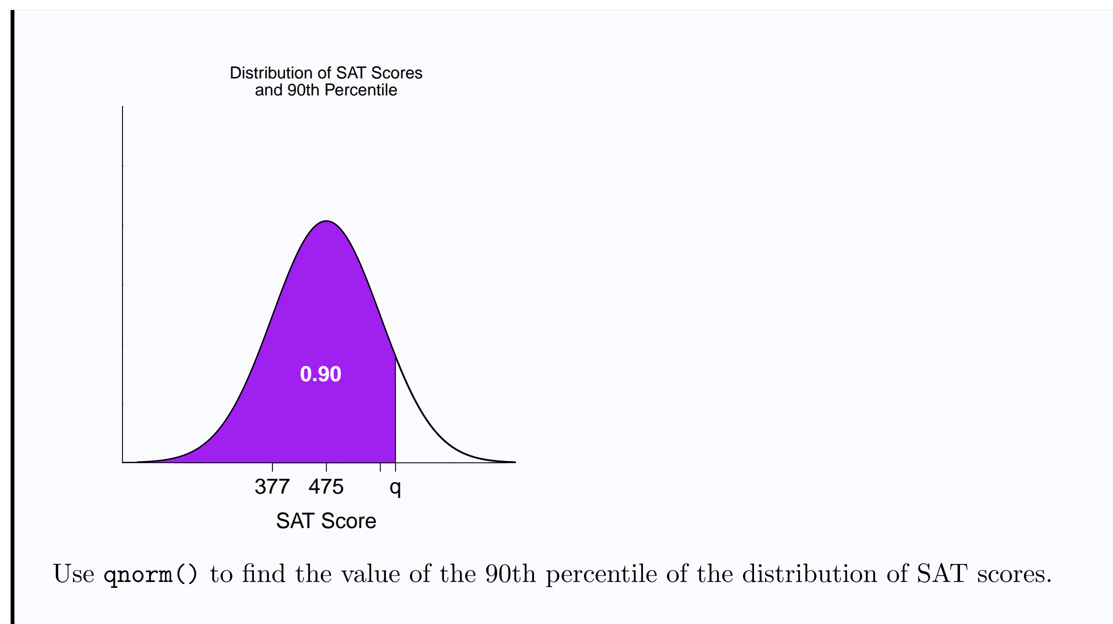


b) What percentage of women are *taller* than 66 inches (i.e. what's the probability that a randomly selected woman's height will be *greater than* than 66 inches)? Use `1 - pnorm()` to find the answer. This is represented by the shaded area below.



Exercise 11 Scores on the verbal Scholastic Aptitude Test (SAT) follow a normal(μ, σ) curve, with mean $\mu = 475$ and standard deviation $\sigma = 98$.

The **90th percentile** of the distribution of SAT scores is the score below which 90% of all scores lie. It's the value marked q on the horizontal axis below.



Use `qnorm()` to find the value of the 90th percentile of the distribution of SAT scores.

1.4 Functions for Other Statistical Distributions

- We can generate a random values, find probabilities, determine percentiles, and evaluate density curves for any of the common discrete or continuous probability distributions. Here's the syntax for the most important ones:

```
r, p, q, or d followed by:

  unif()        # Uniform distribution
  norm()        # Normal distribution
  exp()         # Exponential distribution
  gamma()       # Gamma distribution
  cauchy()      # Cauchy distribution
  lnorm()       # Lognormal distribution
  t()           # T distribution
  f()           # F distribution
  chisq()       # Chi-squared distribution
  binom()       # Binomial distribution
  pois()        # Poisson distribution
  nbinom()      # Negative binomial distribution
  geom()        # Geometric distribution
  hyper()       # Hypergeometric distribution
```

- In each case, the function name is prefaced by `r` to generate a random sample (e.g. `rexp()`), by `p` to find a probability (e.g. `pexp()`), by `q` to determine a percentile (e.g. `qexp()`), and by `d` to evaluate the density function (e.g. `dexp()`).

1.5 Set Operations

- R has several functions that perform ***set operations***:

```
union()       # Union of two vectors x and y, consisting of all
              # the elements that are in at least one of x or y.
intersect()   # Intersection of two vectors x and y, consisting of all
              # the elements that are in both x and y.
setdiff()     # Set difference of two vectors x and y, consisting of
              # all the elements that are in x but not in y.
setequal()    # Returns TRUE or FALSE depending on whether the elements
              # of two vectors x and y are the same.
%in%          # Returns TRUE or FALSE depending on whether a value x
              # is an element of a vector y.  We could also use
              # is.element().
```

- `union()` takes two vectors `x` and `y`, and returns a vector consisting of all the elements that are in `x` ***or*** in `y`. For example:

```
> union(x = 1:5, y = 4:8)            # Which values are in 1:5 or in 4:8?

[1] 1 2 3 4 5 6 7 8
```

- `intersect()` takes two vectors `x` and `y`, and returns a vector consisting of all the elements that are in `x` ***and*** in `y`:

```
> intersect(x = 1:5, y = 4:8)        # Which values are in 1:5 and in 4:8?

[1] 4 5
```

- `setdiff()` takes vectors `x` and `y`, and returns a vector consisting of all the elements that are in `x` ***but not*** in `y`:

```
> setdiff(x = 1:5, y = 4:8)          # Which values are in 1:5 but not in 4:8?

[1] 1 2 3
```

- `setequal()` takes two vectors `x` and `y`, and returns `TRUE` or `FALSE` depending on whether their elements are the ***same***:

```
> setequal(x = c(1, 2, 3), y = c(3, 1, 2))

[1] TRUE
```

- `%in%` Returns `TRUE` or `FALSE` depending on whether a value `x` is ***in*** (i.e. is an ***element of***) a vector `y`. For example:

```
> 3 %in% c(4, 6, 2, 3, 7, 9)
```

```
[1] TRUE
```

Above, we'd get the same result if we typed `is.element(3, c(4, 6, 2, 3, 7, 9))`.

The first argument can be a vector. For example:

```
> c(3, 2) %in% c(4, 6, 2, 3, 7, 9)
```

```
[1] TRUE TRUE
```

Section 1.5 Exercises

Exercise 12 Guess what the result of each of the following commands will be, then check your answers.

a) `> union(x = 1:4, y = 3:6)`

b) `> intersect(x = 1:4, y = 3:6)`

c) `> setdiff(x = 1:4, y = 3:6)`

Exercise 13 Guess what the result of each of the following commands will be, then check your answers.

a) `> setequal(x = c(8, 4, 2), y = c(2, 8, 4))`

b) `> setequal(x = c(4, 7, 9), y = c(7, 9, 6))`

c) `> 4 %in% c(1, 5, 2, 6, 8)`

d) `> 4 %in% c(1, 4, 2, 6, 8)`

e) `> c(2, 4) %in% c(1, 4, 2, 6, 8)`