CSCI E-5a Spring 2022 Midterm Assessment Solutions

Let's clear the global computing environment:

```
rm( list = ls() )
```

Let's load in the Midterm Assessment R objects:

```
load( "Midterm Assessment R Objects.Rdata" )

ls()
```

```
##  [1] "problem.2.exam.1.raw.score.data"
##  [2] "problem.2.exam.2.raw.score.data"
##  [3] "problem.2.paper.raw.score.data"
##  [4] "problem.2.spotify.raw.score.data"
##  [5] "problem.4.data"
##  [6] "problem.5.brand.data"
##  [7] "problem.5.number.of.bottles.sold.data"
##  [8] "problem.7.attempts.data"
##  [9] "problem.7.completions.data"
## [10] "problem.7.interception.data"
## [11] "problem.7.total.yards.data"
## [12] "problem.7.touchdown.pass.data"
## [13] "problem.8.category.data"
## [14] "problem.8.x.data"
## [15] "problem.8.y.data"
```

Problem 1: The Normal Distribution

In Week 5 Module 6: Histograms and Curves we learned how to graph the density function for the normal distribution. Now we're going to investigate the properties of these normal density curves by constructing some comparative visualizations.

Recall that in order to fully specify a normal distribution, we must assign values to two parameters:

- The mean parameter `mean`
- The standard deviation parameter `sd`.

The density curve for a normal distribution has four fundamental properties:

- The curve exhibits a characteristic "mountain" shape and is symmetrical.
- The central location of the density curve is determined by the value of the mean parameter `mean`.
- The width of the density curve is determined by the value of the standard deviation parameter `sd`.
- The area underneath a normal density curve is always 1.

Part (a): Standard normal density curve

The *standard* normal density has a mean of 0 and a standard deviation of 1.

The algebraic form of the standard normal density function is:

$$f(x) \;=\; \frac{1}{\sqrt{2\pi}} \cdot e^{-x^2/2}$$

It's hard to read the tiny print in the exponent, so I prefer to write this formula as:

$$f(x) \;=\; \frac{1}{\sqrt{2\pi}} \cdot \exp\left\{-\frac{x^2}{2}\right\}$$

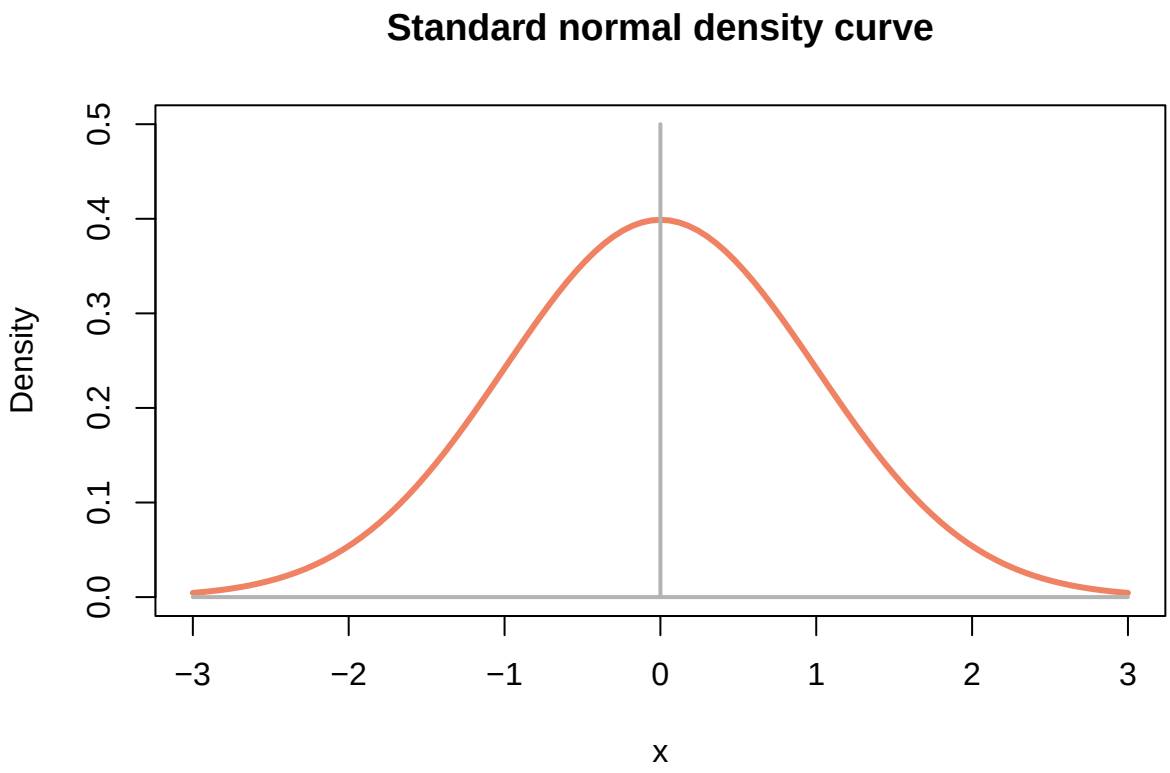
Construct a graph of the standard normal density curve:

- The x-axis should range from -3 to +3.
- The y-axis should range from 0 to 0.5.
- Include a main title, and titles for the x- and y-axes.
- Explicitly specify the line type, width, and color.
- Draw a horizontal reference line $y = 0$ across the range of the x-axis.
- Draw a vertical reference line $x = 0$ across the range of the y-axis.
- Draw the standard normal density curve (i.e. a normal density curve with mean 0 and standard deviation 1) by using the algebraic formula for the standard normal density and the `curve()` function.

Solution

```
curve(
    expr = 1/sqrt(2*pi) * exp( - x^2/2 ),
    xlim = c(-3, +3),
    ylim = c(0, 0.5),
    main = "Standard normal density curve",
    xlab = "x",
    ylab = "Density",
    lty = "solid",
    lwd = 3,
    col = "salmon2"
)

segments(
    x0 = -3,
    y0 = 0,
    x1 = +3,
    y1 = 0,
    lty = "solid",
    lwd = 2,
    col = "gray70"
)

segments(
```

```
    x0 = 0,
    y0 = 0,
    x1 = 0,
    y1 = 0.5,
    lty = "solid",
    lwd = 2,
    col = "gray70"
)
```



Part (b): Location

In this part, we'll graph three normal density curves, varying the mean of the density curve and holding the standard deviation fixed:

- The first normal density curve has a mean of 10 and a standard deviation of 2.5.
- The second normal density curve has a mean of 20 and a standard deviation of 2.5.
- The third normal density curve has a mean of 30 and a standard deviation of 2.5.

To comparatively visualize these different density curves:

- First, construct an empty plot with no data:
  - The x-axis should range from 0 to 40.

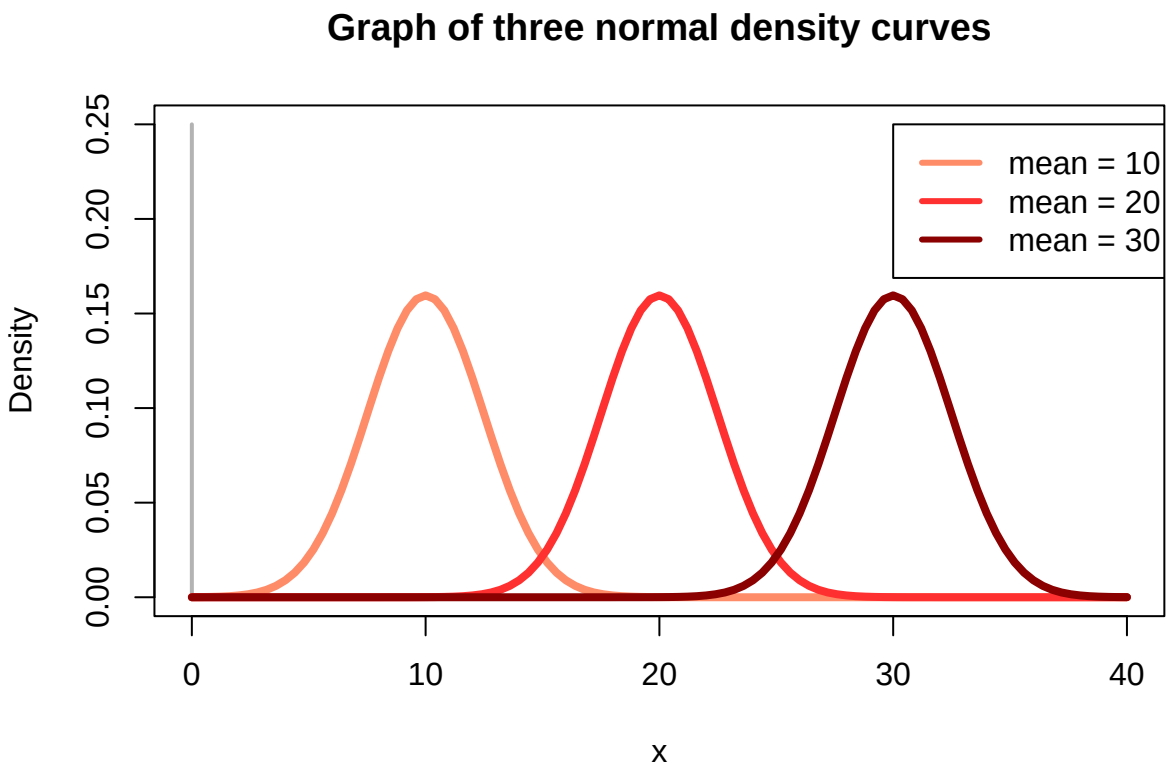
  - The y-axis should range from 0 to 0.25.
  - Include a main title, and titles for the x- and y-axes.

- Draw horizontal and vertical reference lines.
- Draw the density curve for each of the three normal distributions:
  - For the first distribution, use a solid line of width 4 and the color "salmon1".
  - For the second distribution, use a solid line of width 4 and the color "firebrick1".
  - For the third distribution, use a solid line of width 4 and color "darkred".
- Finally, include a legend for the graph, and make sure that the colors for the legend match the colors of the density curves.

Solution

```
plot(
    x = NULL,
    xlim = c(0, 40),
    ylim = c(0, 0.25),
    main = "Graph of three normal density curves",
    xlab = "x",
    ylab = "Density"
)

# Draw the horizontal reference line

segments(
    x0 = 0,
    y0 = 0,
    x1 = 40,
    y1 = 0,
    lty = "solid",
    lwd = 2,
    col = "gray70"
)


# Draw the vertical reference line

segments(
    x0 = 0,
    y0 = 0,
    x1 = 0,
    y1 = 0.25,
    lty = "solid",
    lwd = 2,
    col = "gray70"
)


curve(
    dnorm(x, mean = 10, sd = 2.5),
    lty = "solid",
```

```
    lwd = 4,
    col = "salmon1",
    add = TRUE
)


curve(
    dnorm(x, mean = 20, sd = 2.5),
    lty = "solid",
    lwd = 4,
    col = "firebrick1",
    add = TRUE
)


curve(
    dnorm(x, mean = 30, sd = 2.5),
    lty = "solid",
    lwd = 4,
    col = "darkred",
    add = TRUE
)


legend(
    x = 30,
    y = 0.25,
    legend =
        c(
            "mean = 10",
            "mean = 20",
            "mean = 30"
        ),
    lwd = 3,
    col =
        c(
            "salmon1",
            "firebrick1",
            "darkred"
        )
)
```



Part (c): Width

In this part, we'll graph three normal density curves, holding the mean of the density curve fixed and varying the standard deviation:

- The first normal density curve has a mean of 50 and a standard deviation of 5.
- The second normal density curve has a mean of 50 and a standard deviation of 10.
- The third normal density curve has a mean of 50 and a standard deviation of 15.

To comparatively visualize these different density curves:

- First, construct an empty plot with no data:
  - The x-axis should range from 0 to 100.
  - The y-axis should range from 0 to 0.1.
  - Include a main title, and titles for the x- and y-axes.
- Draw horizontal and vertical reference lines.
- Draw the density curve for each of the three normal distributions:
  - For the first distribution, use a solid line of width 4 and the color "skyblue3".
  - For the second distribution, use a solid line of width 4 and the color "royalblue3".

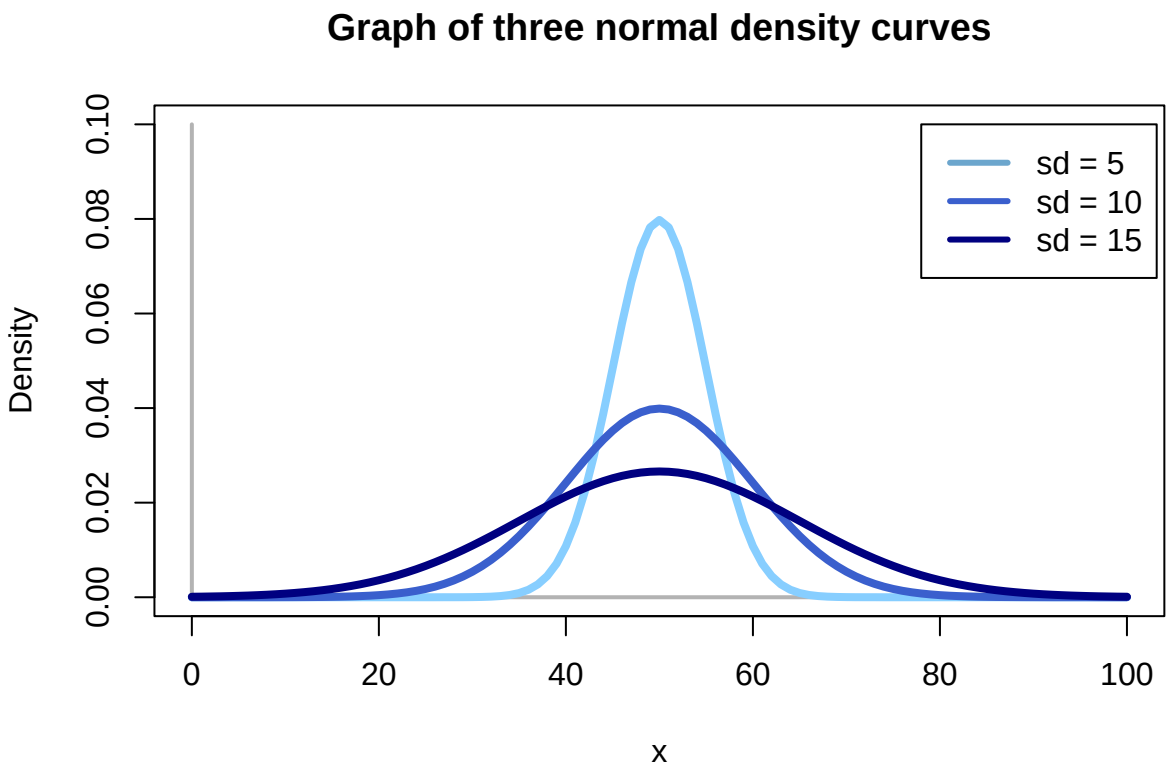
  - For the third distribution, use a solid line of width 4 and color “navy”.

- Finally, include a legend for the graph, and make sure that the colors for the legend match the colors of the density curves.

Solution

```
plot(
    x = NULL,
    xlim = c(0, 100),
    ylim = c(0, 0.1),
    main = "Graph of three normal density curves",
    xlab = "x",
    ylab = "Density"
)

# Draw the horizontal reference line

segments(
    x0 = 0,
    y0 = 0,
    x1 = 100,
    y1 = 0,
    lty = "solid",
    lwd = 2,
    col = "gray70"
)


# Draw the vertical reference line

segments(
    x0 = 0,
    y0 = 0,
    x1 = 0,
    y1 = 0.1,
    lty = "solid",
    lwd = 2,
    col = "gray70"
)


curve(
    dnorm(x, mean = 50, sd = 5),
    lty = "solid",
    lwd = 4,
    col = "skyblue1",
    add = TRUE
)


curve(
    dnorm(x, mean = 50, sd = 10),
    lty = "solid",
```

```
    lwd = 4,
    col = "royalblue3",
    add = TRUE
)


curve(
    dnorm(x, mean = 50, sd = 15),
    lty = "solid",
    lwd = 4,
    col = "navy",
    add = TRUE
)


legend(
    x = 78,
    y = 0.1,
    legend =
        c(
            "sd = 5",
            "sd = 10",
            "sd = 15"
        ),
    lwd = 3,
    col =
        c(
            "skyblue3",
            "royalblue3",
            "navy"
        )
)
```



Part (d): Mixture

In this part, we'll graph three normal density curves, varying both the mean and standard deviation:

- The first normal density curve is a standard normal density with a mean of 0 and a standard deviation of 1.
- The second normal density curve has a mean of 2 and a standard deviation of 0.5.
- The third normal density curve has a mean of 5 and a standard deviation of 2.

To comparatively visualize these different density curves:

- First, construct an empty plot with no data:
  - The x-axis should range from -5 to 10.
  - The y-axis should range from 0 to 1.
  - Include a main title, and titles for the x- and y-axes.
- Draw horizontal and vertical reference lines.
- Draw the density curve for each of the three normal distributions:
  - For the first distribution, use a solid line of width 4 and the color "plum2".
  - For the second distribution, use a solid line of width 4 and the color "darkorchid3".

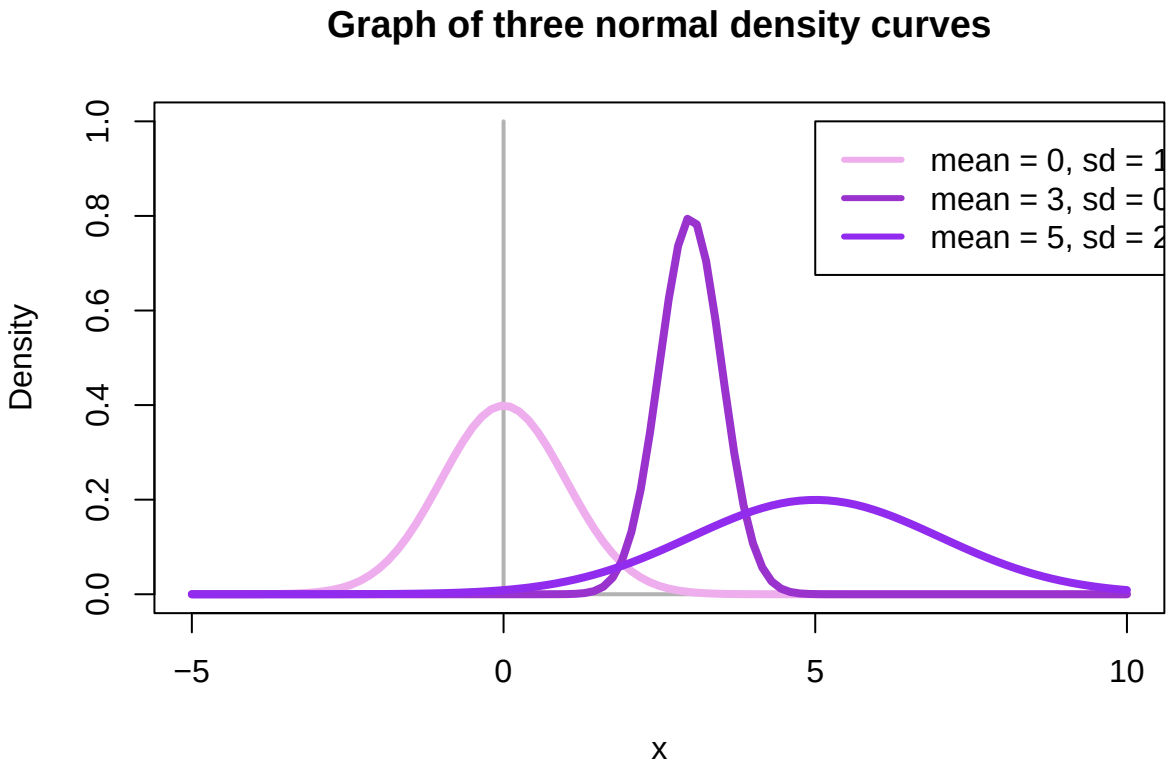
  - For the third distribution, use a solid line of width 4 and color “purple2”.

- Finally, include a legend for the graph, and make sure that the colors for the legend match the colors of the density curves.

Solution

```
plot(
    x = NULL,
    xlim = c(-5, 10),
    ylim = c(0, 1),
    main = "Graph of three normal density curves",
    xlab = "x",
    ylab = "Density"
)

# Draw the horizontal reference line

segments(
    x0 = -5,
    y0 = 0,
    x1 = 10,
    y1 = 0,
    lty = "solid",
    lwd = 2,
    col = "gray70"
)


# Draw the vertical reference line

segments(
    x0 = 0,
    y0 = 0,
    x1 = 0,
    y1 = 1,
    lty = "solid",
    lwd = 2,
    col = "gray70"
)


curve(
    dnorm(x, mean = 0, sd = 1),
    lty = "solid",
    lwd = 4,
    col = "plum2",
    add = TRUE
)


curve(
    dnorm(x, mean = 3, sd = 0.5),
    lty = "solid",
```

```
    lwd = 4,
    col = "darkorchid3",
    add = TRUE
)


curve(
    dnorm(x, mean = 5, sd = 2),
    lty = "solid",
    lwd = 4,
    col = "purple2",
    add = TRUE
)


legend(
    x = 5,
    y = 1,
    legend =
        c(
            "mean = 0, sd = 1",
            "mean = 3, sd = 0.5",
            "mean = 5, sd = 2"
        ),
    lwd = 3,
    col =
        c(
            "plum2",
            "darkorchid3",
            "purple2"
        )
)
```


Graph of three normal density curves
Density
0.0
0.2
0.4
0.6
0.8
1.0
−5
0
5
10
x
mean = 0, sd = 1
mean = 3, sd = 0
mean = 5, sd = 2

End of Problem 1

Problem 2: Final Grades

The grading system for the course "Contemporary Concepts in Modern Society" has three components:

- Exam 1
- Exam 2
- The final project, which is either a published paper in a peer-reviewed academic journal or a curated Spotify playlist.

Part (a): Exam 1 standardized scores

The vector `problem.2.exam.1.raw.score.data` contains the raw scores for Exam 1.

Standardize the Exam 1 scores so that a raw score of 65 points corresponds to a standardized score of 100, and save these standardized scores in a variable.

Then report the first five values of the standardized scores by using a `cat()` statement, displaying the values with 2 decimal places.

Solution

```
exam.1.standardized.score.vector <-
    problem.2.exam.1.raw.score.data / 65 * 100

cat(
    "Exam 1 standardized scores:",
    formatC(
        head( exam.1.standardized.score.vector, n = 5 ),
        format = "f",
        digits = 2
    ),
    "\n"
)
```

```
## Exam 1 standardized scores: 95.38 78.46 96.92 86.15 78.46
```

Part (b): Exam 2 standardized scores

The vector `problem.2.exam.2.raw.score.data` contains the raw scores for Exam 2.

Standardize the Exam 2 scores so that a raw score of 65 points corresponds to a standardized score of 100, and save these standardized scores in a variable.

Then report the first five values of the standardized scores by using a `cat()` statement, displaying the values with 2 decimal places.

Solution

```
exam.2.standardized.score.vector <-
    problem.2.exam.2.raw.score.data / 65 * 100

cat(
    "Exam 2 standardized scores:",
```

```
    formatC(
        head( exam.2.standardized.score.vector, n = 5 ),
        format = "f",
        digits = 2
    ),
    "\n"
)
```

```
## Exam 2 standardized scores: 95.38 89.23 92.31 87.69 70.77
```

Part (c): Published paper standardized scores

The vector `problem.2.paper.raw.score.data` contains the raw scores for the paper published in a peer-reviewed academic journal.

If a student does not submit a paper published in a peer-reviewed academic journal, this is recorded as `NA`, and these values should be converted to a 0.

Standardize the published paper scores so that a raw score of 50 points corresponds to a standardized score of 100, and save these standardized scores in a variable.

Then report the first five values of the standardized scores by using a `cat()` statement, displaying the values with 2 decimal places.

Solution

```
paper.standardized.score.vector <-
    problem.2.paper.raw.score.data / 50 * 100

paper.standardized.score.vector[
    is.na( paper.standardized.score.vector ) ] <-
    0

cat(
    "Paper standardized scores:",
    formatC(
        head( paper.standardized.score.vector, n = 5 ),
        format = "f",
        digits = 2
    ),
    "\n"
)
```

```
## Paper standardized scores: 96.00 78.00 98.00 0.00 74.00
```

Part (d): Spotify playlist standardized scores

The vector `problem.2.spotify.raw.score.data` contains the raw scores for the curated Spotify playlist.

If a student does not submit a curated Spotify playlist, this is recorded as `NA`, and these values should be converted to a 0.

Standardize the Spotify playlist scores so that a raw score of 80 points corresponds to a standardized score of 100, and save these standardized scores in a variable.

Then report the first five values of the standardized scores by using a `cat()` statement, displaying the values with 2 decimal places.

Solution

```
spotify.standardized.score.vector <-
    problem.2.spotify.raw.score.data / 80 * 100

spotify.standardized.score.vector[
    is.na( spotify.standardized.score.vector ) ] <-
    0

cat(
    "Spotify standardized scores:",
    formatC(
        head( spotify.standardized.score.vector, n = 5 ),
        format = "f",
        digits = 2
    ),
    "\n"
)
```

```
## Spotify standardized scores: 92.50 82.50 92.50 81.25 80.00
```

Part (e): Final project standardized score

For each student, the final project standardized score is defined as the maximum of the published paper standard score and the Spotify playlist standardized score.

Calculate the final project standardized scores for the full class of students, and store them in a vector.

Then report the first five values of the standardized final project scores by using a `cat()` statement, displaying the values with 2 decimal places.

Solution

```
final.project.standardized.score.vector <-
    ifelse(
        paper.standardized.score.vector > spotify.standardized.score.vector,
        paper.standardized.score.vector,
        spotify.standardized.score.vector
    )

cat(
    "Final project standardized scores:",
    formatC(
        head( final.project.standardized.score.vector, n = 5 ),
        format = "f",
        digits = 2
    ),
    "\n"
)
```

```
## Final project standardized scores: 96.00 82.50 98.00 81.25 80.00
```

Part (f): Final course score

The final course score is computed as a weighted average of the standardized scores from part (a), (b), and (e):

Component	Weight
Exam 1	0.25
Exam 2	0.25
Final Project	0.50

Compute the final course scores for each student, and save them in a variable. Then display the first 5 values of these final course scores by using a `cat()` statement, formatting the values with 2 decimal places.

Solution

```
final.course.score.vector <-
    0.25 * exam.1.standardized.score.vector +
    0.25 * exam.2.standardized.score.vector +
    0.50 * final.project.standardized.score.vector

cat(
    "Final course score:",
    formatC(
        x = head( final.course.score.vector, n = 5),
        format = "f",
        digits = 2
    ),
    "\n"
)
```

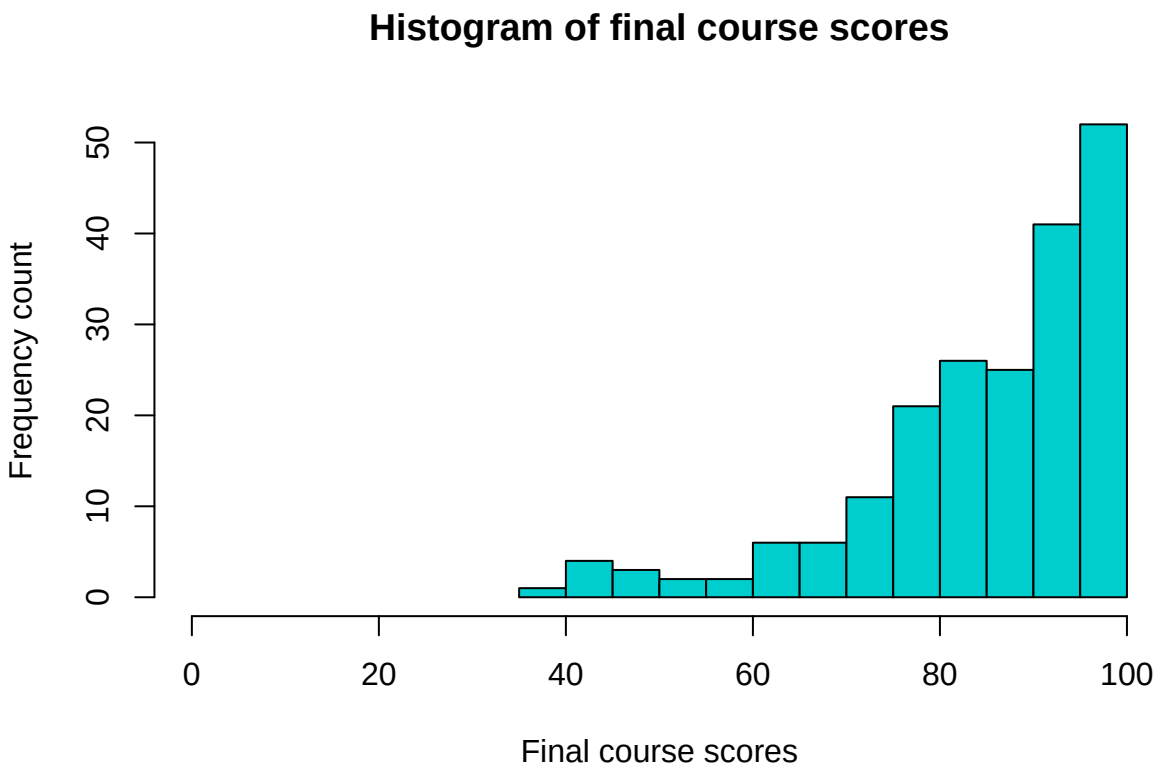
```
## Final course score: 95.69 83.17 96.31 84.09 77.31
```

Part (g): Histogram

Construct a histogram of the final course scores. Set the range of the x-axis to go from 0 to 100, and include a main title and titles for the x- and y-axes. Use 20 breaks and explicitly specify a color for the bars.

Solution

```
hist(
    x = final.course.score.vector,
    xlim = c(0, 100),
    main = "Histogram of final course scores",
    xlab = "Final course scores",
    ylab = "Frequency count",
    col = "cyan3",
    breaks = 20
)
```


Histogram of final course scores
Frequency count
0
10
20
30
40
50
0
20
40
60
80
100
Final course scores

End of Problem 2

Problem 3: Savings Plan

In Week 4, Module 4, we studied the process of loan amortization, in which an initial loan balance is reduced to 0 by making annual payments. Now we'll study the opposite process, called a "savings plan", where we start with a balance of 0 and then make annual deposits into an account with a fixed interest rate.

For instance, suppose Ted has a savings plan where he deposits $1,000 at the beginning of each year, and earns 4% annual interest on his total balance.

- At the beginning of Year 1 before the first deposit the balance of the plan is $0.
- At the beginning of Year 1 after the first deposit the balance of the plan is $1,000.
- At the end of Year 1 after the annual interest has accrued the balance is $1,040.
- In Year 2, the balance at the end of Year 1 becomes the balance at the beginning of Year 2 before the second deposit, and the process continues.

Part (a): Final balance formula

Suppose we make an annual deposit D into a savings plan at the beginning of each year for n years with a fixed interest rate r. Recall the definition of the discount factor ν:

$$\nu = \frac{1}{1+r}$$

Then the final balance of the savings plan at the end of year n, denoted B, is:

$$B = \frac{(1-\nu^n)\cdot D}{r\cdot \nu^{n+1}}$$

Ted deposits $1,000 into a savings plan for 5 years with a fixed interest rate of 4%. Calculate the final balance in the savings plan at the end of the fifth year, and report your result by using a `cat()` statement, formatting the dollar amount with two decimal places.

Solution

```
annual.deposit.amount <- 1000

interest.rate <- 0.04

number.of.payments <- 5
```

```
discount.factor <-
    1 / (1 + interest.rate)

cat(
    "Discount factor:",
    discount.factor,
    "\n"
)
```

```
## Discount factor: 0.9615385
```

```r
final.balance <-
    annual.deposit.amount *
    (1 - discount.factor^number.of.payments) /
    (interest.rate * discount.factor^(number.of.payments + 1) )

cat(
    "Final balance: $",
    formatC(
        final.balance,
        format = "f",
        digits = 2,
        big.mark = ","
    ),
    "\n",
    sep = ""
)
```

```
## Final balance: $5,632.98
```

Part (b): Annual savings report

Ted will make 5 annual deposits of $1,000 at the beginning of each year. The interest rate is 4%.

Print out a report of Ted's savings plan.

- Print out the year, along with its number (i.e. Year 1, Year 2, etc.).
- For each year, print out on a separate line:
  - The balance of the savings plan at the beginning of the year, before Ted makes an annual deposit of $1,000.
  - The balance of the savings plan at the beginning of the year, after Ted makes an annual deposit of $1,000.
  - The balance of the savings plan at the end of the year, after one year's interest has accrued on the savings.

The balance amount at the end of the year becomes the balance amount at the beginning of the next year, before Ted makes the annual deposit.

When you print out the values you should annotate them so that we know what it is that you are reporting; don't just print out the numbers without any explanation.

IMPORTANT: You MUST use a `for()` loop for this problem – do not create a bunch of dedicated variables and then manually write out all the code to update them.

Solution

```r
current.balance <- 0

for( current.year in 1:number.of.payments ) {

    cat( "Current year:", current.year, "\n" )

    cat(
```

```
        "\tCurrent balance at start of year, before deposit:",
        current.balance,
        "\n"
    )

    current.balance <-
        current.balance + annual.deposit.amount

    cat(
        "\tCurrent balance at start of year, after deposit:",
        current.balance,
        "\n"
    )

    current.balance <-
        current.balance + interest.rate * current.balance

    cat(
        "\tCurrent balance at end of year:",
        current.balance,
        "\n\n"
    )
}
```

```
## Current year: 1
##  Current balance at start of year, before deposit: 0
##  Current balance at start of year, after deposit: 1000
##  Current balance at end of year: 1040
##
## Current year: 2
##  Current balance at start of year, before deposit: 1040
##  Current balance at start of year, after deposit: 2040
##  Current balance at end of year: 2121.6
##
## Current year: 3
##  Current balance at start of year, before deposit: 2121.6
##  Current balance at start of year, after deposit: 3121.6
##  Current balance at end of year: 3246.464
##
## Current year: 4
##  Current balance at start of year, before deposit: 3246.464
##  Current balance at start of year, after deposit: 4246.464
##  Current balance at end of year: 4416.323
##
## Current year: 5
##  Current balance at start of year, before deposit: 4416.323
##  Current balance at start of year, after deposit: 5416.323
##  Current balance at end of year: 5632.975
```

Part (c): Annual deposit amount

In part (a), we saw a formula that enables us to calculate the final balance B at the end of n years, given the annual deposit amount D and the annual interest rate r:

$$B = \frac{(1 - \nu^n) \cdot D}{r \cdot \nu^{n+1}}$$

We can solve this formula for D in terms of B, n, and r:

$$D = \frac{r \cdot \nu^{n+1} \cdot B}{1 - \nu^n}$$

Taylor is designing a saving plan so that the balance of the account at the end of 12 years will be exactly \$100,000. The annual interest rate is 2.5%.

Calculate the required annual deposit amount D for Taylor's plan. Report your result by using a `cat()` statement, formatting the dollar amount with two decimal places.

Solution

```
taylor.final.balance.amount <- 100000

taylor.interest.rate <- 0.025

taylor.number.of.payments <- 12
```

```
taylor.discount.factor <-
    1 / (1 + taylor.interest.rate)
```

```
taylor.annual.deposit.amount <-
    taylor.interest.rate *
    taylor.discount.factor^( taylor.number.of.payments + 1 ) *
    taylor.final.balance.amount /
    (1 - taylor.discount.factor^taylor.number.of.payments)
```

```
cat(
    "Annual deposit amount: $",
    formatC(
        taylor.annual.deposit.amount,
        format = "f",
        digits = 2,
        big.mark = ","
    ),
    "\n",
    sep = ""
)
```

```
## Annual deposit amount: $7,071.91
```

Part (d): Savings plan schedule

Taylor is designing a saving plan so that the balance of the account at the end of 12 years will be exactly \$100,000. The annual interest rate is 2.5%.

In part (c), you calculated the annual deposit amount D that Taylor needs in order to obtain the final balance of $100,000.

Print out a report of Taylor's savings plan.

- Print out the year, along with its number (i.e. Year 1, Year 2, etc.).
- For each year, print out on a separate line:
  - The balance of the savings plan at the beginning of the year, before Taylor makes an annual deposit with the amount that you calculated in part (c).
  - The balance of the savings plan at the beginning of the year, after Taylor makes an annual deposit with the amount that you calculated in part (c).
  - The balance of the savings plan at the end of the year, after one year's interest has accrued on the savings.

The balance amount at the end of the year becomes the balance amount at the beginning of the next year, before Taylor makes the annual deposit.

When you print out the values you should annotate them so that we know what it is that you are reporting; don't just print out the numbers without any explanation.

IMPORTANT: You MUST use a `for()` loop for this problem – do not create a bunch of dedicated variables and then manually write out all the code to update them.

Solution

```
taylor.current.balance <- 0

for( current.year in 1:taylor.number.of.payments ) {

    cat( "Current year:", current.year, "\n" )

    cat(
        "\tCurrent balance at start of year, before deposit:",
        formatC(
            taylor.current.balance,
            format = "f",
            digits = 2,
            big.mark = ","
        ),
        "\n"
    )

    taylor.current.balance <-
        taylor.current.balance + taylor.annual.deposit.amount

    cat(
        "\tCurrent balance at start of year, after deposit:",
        formatC(
            taylor.current.balance,
            format = "f",
            digits = 2,
            big.mark = ","
        ),
```

```
        "\n"
    )

    taylor.current.balance <-
        taylor.current.balance + taylor.interest.rate * taylor.current.balance

    cat(
        "\tCurrent balance at end of year:",
        formatC(
            taylor.current.balance,
            format = "f",
            digits = 2,
            big.mark = ","
        ),
        "\n\n"
    )
}
```

```
## Current year: 1
##  Current balance at start of year, before deposit: 0.00
##  Current balance at start of year, after deposit: 7,071.91
##  Current balance at end of year: 7,248.71
##
## Current year: 2
##  Current balance at start of year, before deposit: 7,248.71
##  Current balance at start of year, after deposit: 14,320.63
##  Current balance at end of year: 14,678.64
##
## Current year: 3
##  Current balance at start of year, before deposit: 14,678.64
##  Current balance at start of year, after deposit: 21,750.56
##  Current balance at end of year: 22,294.32
##
## Current year: 4
##  Current balance at start of year, before deposit: 22,294.32
##  Current balance at start of year, after deposit: 29,366.24
##  Current balance at end of year: 30,100.39
##
## Current year: 5
##  Current balance at start of year, before deposit: 30,100.39
##  Current balance at start of year, after deposit: 37,172.31
##  Current balance at end of year: 38,101.62
##
## Current year: 6
##  Current balance at start of year, before deposit: 38,101.62
##  Current balance at start of year, after deposit: 45,173.53
##  Current balance at end of year: 46,302.87
##
## Current year: 7
##  Current balance at start of year, before deposit: 46,302.87
##  Current balance at start of year, after deposit: 53,374.78
##  Current balance at end of year: 54,709.15
##
```

```
## Current year: 8
##  Current balance at start of year, before deposit: 54,709.15
##  Current balance at start of year, after deposit: 61,781.07
##  Current balance at end of year: 63,325.59
##
## Current year: 9
##  Current balance at start of year, before deposit: 63,325.59
##  Current balance at start of year, after deposit: 70,397.51
##  Current balance at end of year: 72,157.45
##
## Current year: 10
##  Current balance at start of year, before deposit: 72,157.45
##  Current balance at start of year, after deposit: 79,229.36
##  Current balance at end of year: 81,210.10
##
## Current year: 11
##  Current balance at start of year, before deposit: 81,210.10
##  Current balance at start of year, after deposit: 88,282.01
##  Current balance at end of year: 90,489.06
##
## Current year: 12
##  Current balance at start of year, before deposit: 90,489.06
##  Current balance at start of year, after deposit: 97,560.98
##  Current balance at end of year: 100,000.00
```

End of Problem 3

Problem 4: Experimental Data

The vector `problem.4.data` contains simulated laboratory measurements of an enzyme for a clinical trial, which on average are about 1.3 with a standard deviation of 0.2. However, some of the values are not within this range:

- If a subject has dropped out of the study, the measurement is recorded as -99.
- If a subject is still an active participant in this study, but missed the laboratory appointment, the measurement is recorded as -98.
- Sometimes the enzyme assay malfunctions and produces an extremely high value for the measurement.

Our goal in this problem is to summarize the values that represent a valid measurement, and to visualize the distribution of these values.

Part (a): Sample mean and standard deviation

Calculate the sample mean of the data in `problem.4.data`, and report your result using a `cat()` statement, formatting the value with 2 decimal places.

Then calculate the sample standard deviation of the data in `problem.4.data`, and report your result by using a `cat()` statement, formatting the value with 2 decimal places.

Solution

```
sample.mean <-
    mean( problem.4.data )

cat(
    "Sample mean of problem.4.data:",
    formatC(
        sample.mean,
        format = "f",
        digits = 2
    ),
    "\n"
)
```

```
## Sample mean of problem.4.data: -0.45
```

```
sample.standard.deviation <-
    sd( problem.4.data, na.rm = TRUE )

cat(
    "Sample standard deviation of problem.4.data:",
    formatC(
        sample.standard.deviation,
        format = "f",
        digits = 2
    ),
    "\n"
)
```

```
## Sample standard deviation of problem.4.data: 16.86
```

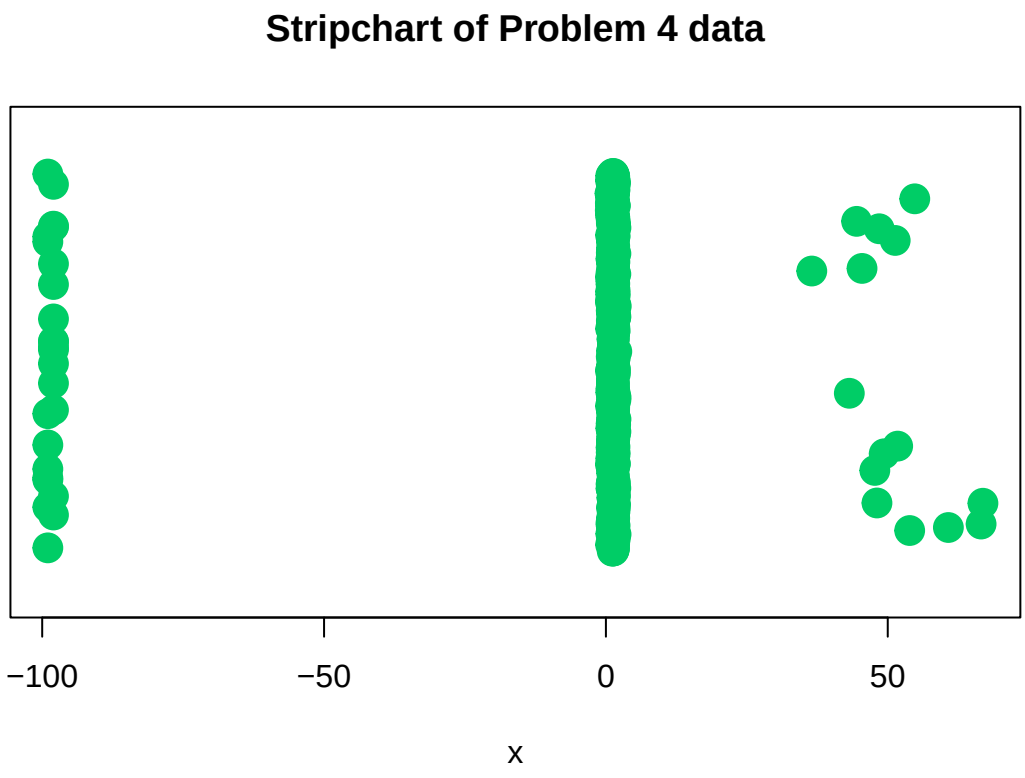
Part (b): Stripchart

Construct a stripchart of the data in `problem.4.data`. Do not specify the range of the x-axis; instead, let R determine the range based on the observed data.

Be sure to include a main title and a title for the x-axis, and explicitly specify the shape, size, and color of the points. Use jitter with a jitter amount of 0.8.

Solution

```
stripchart(
    problem.4.data,
    ylim = c(0, 2),
    main = "Stripchart of Problem 4 data",
    xlab = "x",
    pch = 19,
    cex = 2,
    col = "springgreen3",
    method = "jitter",
    jitter = 0.8
)
```

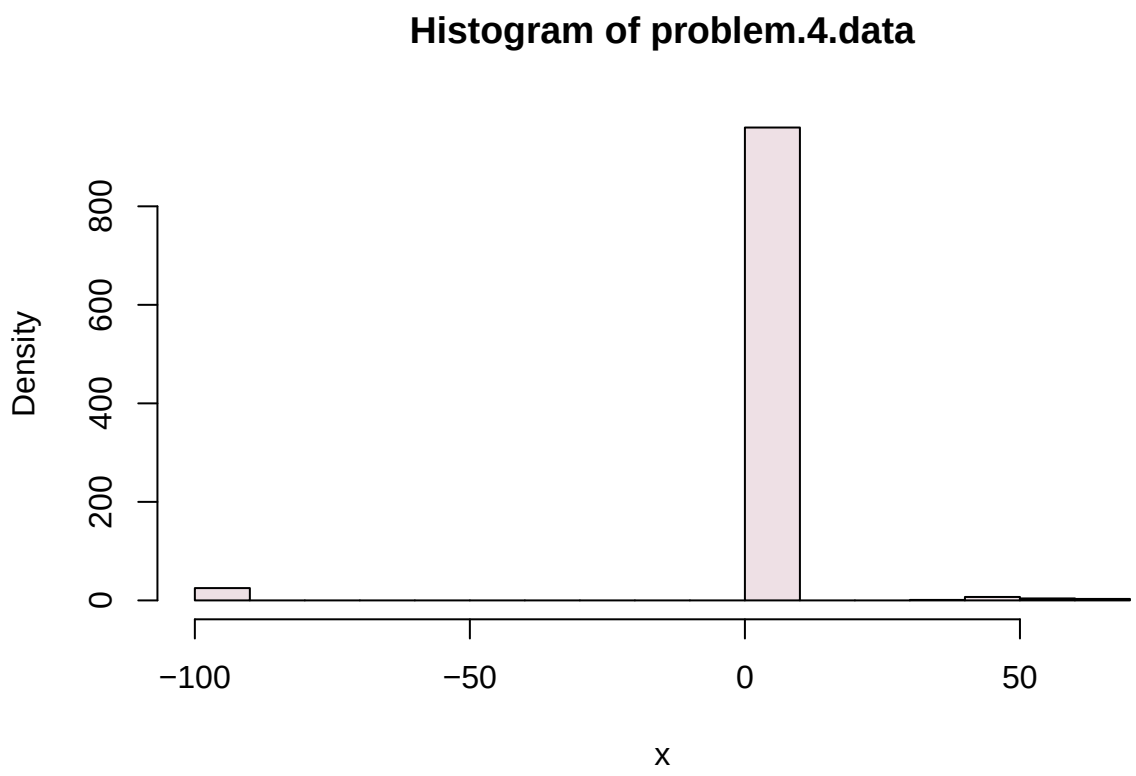


Part (c): Histogram

Construct a histogram of the raw data in `problem.4.data`. Do not specify the range of the x-axis; instead, let R determine the range based on the observed data. Include a main title and titles for the x- and y-axes. Use 20 breaks and explicitly specify a color for the bars.

Solution

```
hist(
    x = problem.4.data,
    main = "Histogram of problem.4.data",
    xlab = "x",
    ylab = "Density",
    col = "lavenderblush2",
    breaks = 20
)
```



Part (d): -99 values

How many subjects in the study have dropped out? Determine this answer by counting the number of -99 values in `problem.4.data`, and report your answer by using a `cat()` statement, formatting the value with 0 decimal places.

Then use a `for()` loop to generate a short report listing all the locations in `problem.4.data` which contain a -99.

Solution

```
number.of.minus.99.values <-
    sum( problem.4.data == -99 )

cat( "The number of -99 values:", number.of.minus.99.values, "\n" )
```

```
## The number of -99 values: 10
```

```
minus.99.locations <-
    which( problem.4.data == -99 )

for( i in minus.99.locations ) {
    cat(
        "-99 value in location:",
        i,
        "\n"
    )
}
```

```
## -99 value in location: 31
## -99 value in location: 161
## -99 value in location: 292
## -99 value in location: 329
## -99 value in location: 385
## -99 value in location: 409
## -99 value in location: 454
## -99 value in location: 774
## -99 value in location: 932
## -99 value in location: 933
```

Part (e): Repairing the -99 values

Repair the -99 values in `problem.4.data` by replacing them with `NA` values. Then, to check that your code is correct, count the number of elements in the repaired vector that are equal to -99, and report this value using a `cat()` statement, displaying the value with 0 decimal places.

Solution

```
problem.4.data[ problem.4.data == -99 ] <- NA
```

```
number.of.minus.99.values <-
    sum( problem.4.data == -99, na.rm = TRUE )

cat( "The number of -99 values:", number.of.minus.99.values, "\n" )
```

```
## The number of -99 values: 0
```

Part (f): -98 values

How many subjects are still active participants in the study, but missed the laboratory appointment? Determine this answer by counting the number of -98 values in the repaired data from part (c), and report your answer by using a `cat()` statement, formatting the value with 0 decimal places.

Then use a `for()` loop to generate a short report listing all the locations in `problem.4.data` which contain a -98.

Solution

```
number.of.minus.98.values <-
    sum( problem.4.data == -98, na.rm = TRUE )

cat( "The number of -98 values:", number.of.minus.98.values, "\n" )
```

```
## The number of -98 values: 15
```

```
minus.98.locations <-
    which( problem.4.data == -98 )

for( i in minus.98.locations ) {
    cat(
        "-98 value in location:",
        i,
        "\n"
    )
}
```

```
## -98 value in location: 56
## -98 value in location: 104
## -98 value in location: 120
## -98 value in location: 146
## -98 value in location: 182
## -98 value in location: 328
## -98 value in location: 404
## -98 value in location: 411
## -98 value in location: 502
## -98 value in location: 665
## -98 value in location: 750
## -98 value in location: 842
## -98 value in location: 948
## -98 value in location: 974
## -98 value in location: 978
```

Part (g): Repairing the -98 values

Repair the -98 values in `problem.4.data` by replacing them with `NA` values. Then, to check that your code is correct, count the number of elements in the repaired vector that are equal to -98, and report this value using a `cat()` statement, displaying the value with 0 decimal places.

Solution

```
problem.4.data[ problem.4.data == -98 ] <- NA
```

```
number.of.minus.98.values <-
    sum( problem.4.data == -98, na.rm = TRUE )

cat( "The number of -98 values:", number.of.minus.98.values, "\n" )
```

```
## The number of -98 values: 0
```

Part (h): Malfunctions

If an enzyme measurement is strictly greater than 5, it is considered to be a malfunction. How many enzyme measurements in the data were malfunctions? Determine this answer by counting the number of values in `problem.4.data` that are strictly greater than 5, and report your answer by using a `cat()` statement, formatting the value with 0 decimal places.

Then use a `for()` loop to generate a short report listing all the locations in `problem.4.data` which contain a value that is strictly greater than 5.

Solution

```
number.of.malfunctions <-
    sum( problem.4.data > 5, na.rm = TRUE )

cat( "The number of malfunctions:", number.of.malfunctions, "\n" )
```

```
## The number of malfunctions: 15
```

```
malfunction.locations <-
    which( problem.4.data > 5 )

for( i in malfunction.locations ) {
    cat(
        "Malfunction in location:",
        i,
        "\n"
    )
}
```

```
## Malfunction in location: 78
## Malfunction in location: 81
## Malfunction in location: 110
## Malfunction in location: 166
## Malfunction in location: 174
## Malfunction in location: 184
## Malfunction in location: 284
## Malfunction in location: 303
## Malfunction in location: 372
## Malfunction in location: 414
## Malfunction in location: 490
## Malfunction in location: 611
## Malfunction in location: 793
## Malfunction in location: 806
## Malfunction in location: 984
```

Part (i): Repairing the malfunction values

Repair the malfunction values in `problem.4.data` by replacing them with `NA` values. Then, to check that your code is correct, count the number of elements in the repaired vector that are malfunctions, and report this value using a `cat()` statement, displaying the value with 0 decimal places.

Solution

```
problem.4.data[ problem.4.data > 5 ] <- NA
```

```
number.of.malfunction.values <-
    sum( problem.4.data > 5, na.rm = TRUE )

cat( "The number of malfunction values:", number.of.malfunction.values, "\n" )
```

```
## The number of malfunction values: 0
```

Part (j): Sample mean and standard deviation

Calculate the sample mean of the data in `problem.4.data`, and report your result using a `cat()` statement, formatting the value with 2 decimal places.

Then calculate the sample standard deviation of the data in `problem.4.data`, and report your result using a `cat()` statement, formatting the value with 2 decimal places.

Solution

```
sample.mean <-
    mean( problem.4.data, na.rm = TRUE )

cat(
    "Sample mean of problem.4.data:",
    formatC(
        sample.mean,
        format = "f",
        digits = 2
    ),
    "\n"
)
```

```
## Sample mean of problem.4.data: 1.29
```

```
sample.standard.deviation <-
    sd( problem.4.data, na.rm = TRUE )

cat(
    "Sample standard deviation of problem.4.data:",
    formatC(
        sample.standard.deviation,
        format = "f",
        digits = 2
    ),
    "\n"
)
```

```
## Sample standard deviation of problem.4.data: 0.19
```

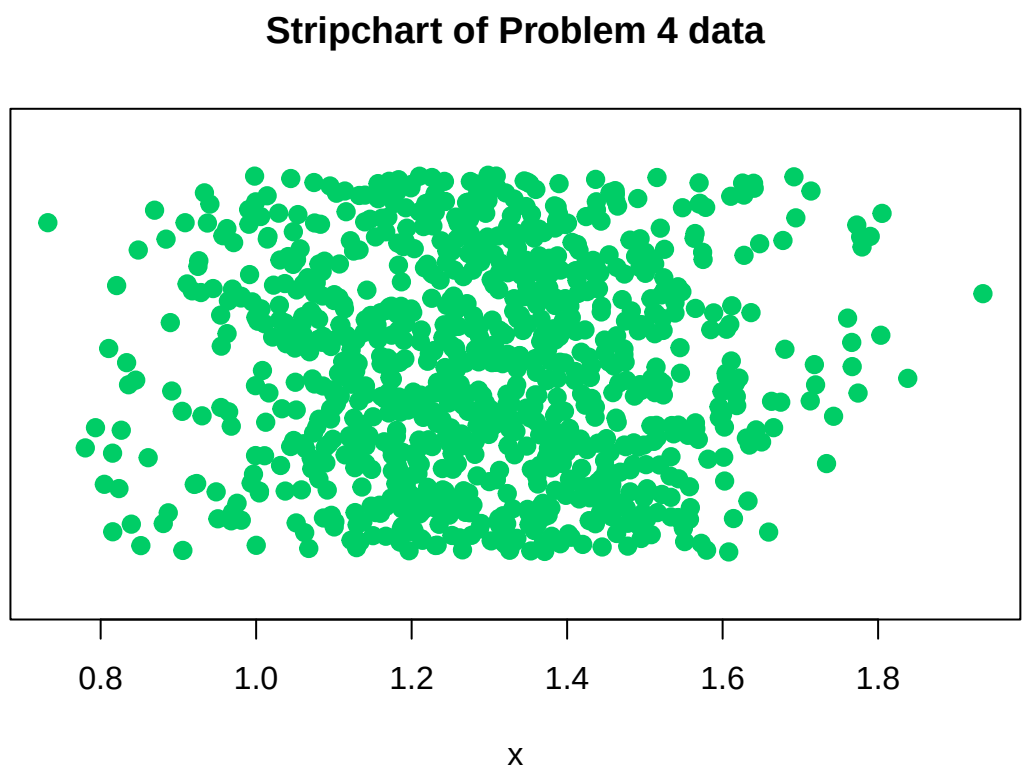
Part (k): Stripchart

Construct a stripchart of the data in `problem.4.data`. Do not specify the range of the x-axis; instead, let R determine the range based on the observed data.

Be sure to include a main title and a title for the x-axis, and explicitly specify the shape, size, and color of the points. Use jitter with a jitter amount of 0.8.

Solution

```
stripchart(
    problem.4.data,
    ylim = c(0, 2),
    main = "Stripchart of Problem 4 data",
    xlab = "x",
    pch = 19,
    cex = 1.2,
    col = "springgreen3",
    method = "jitter",
    jitter = 0.8
)
```

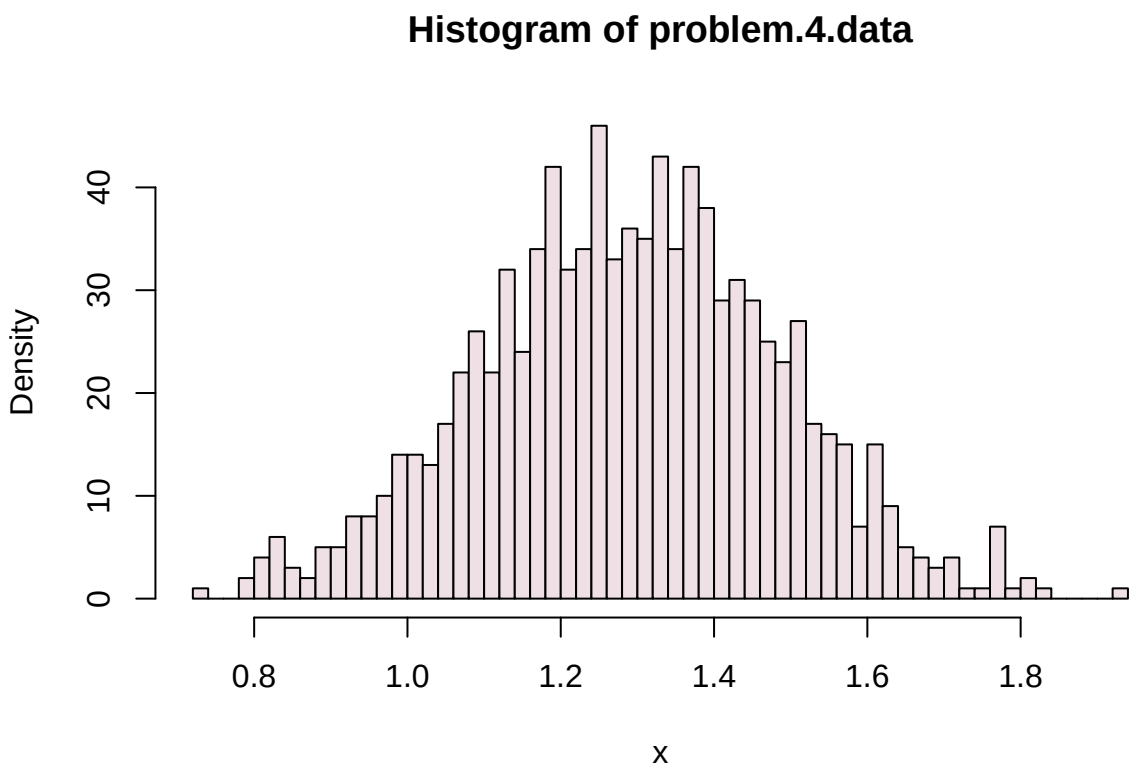


Part (l): Histogram

Construct a histogram of the fully cleaned data from part (i). Be sure to include a main title and titles for the x- and y-axes. Don't specify a range for the x-axis or y-axis, but instead allow R to make this determination. Specify a color for the bars of the histogram and use 50 breaks.

Solution

```
hist(
    x = problem.4.data,
    main = "Histogram of problem.4.data",
    xlab = "x",
    ylab = "Density",
    col = "lavenderblush2",
    breaks = 50
)
```



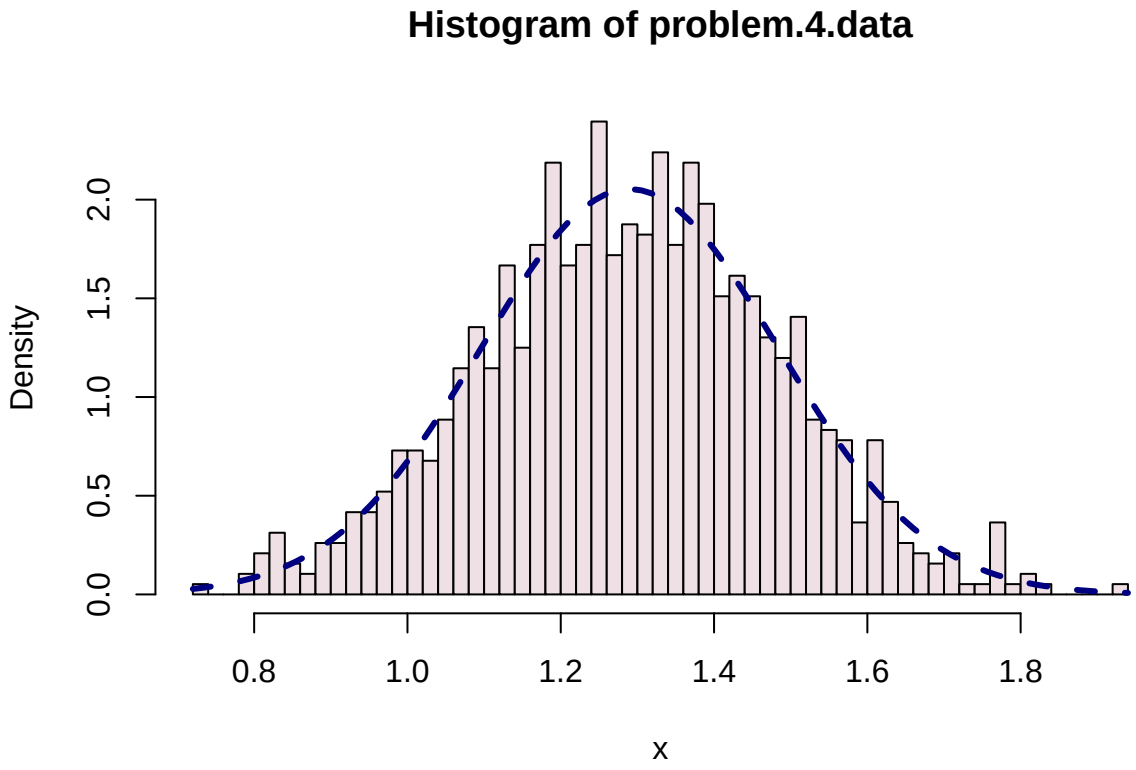
Part (m): Histogram with normal density curve

Copy your code from part (l) for creating a histogram of the fully cleaned data from part (i). Then superimpose the best-fitting normal density curve for this data. Be sure to specify the line type, width, and color for the density curve.

Solution

```
sample.mean <-
    mean( problem.4.data, na.rm = TRUE )

sample.sd <-
    sd( problem.4.data, na.rm = TRUE )
```

```
hist(
    x = problem.4.data,
    main = "Histogram of problem.4.data",
    xlab = "x",
    ylab = "Density",
    prob = TRUE,
    col = "lavenderblush2",
    breaks = 50
)

curve(
    dnorm(x, mean = sample.mean, sd = sample.sd ),
    lwd = 3,
    lty = "dashed",
    col = "navy",
    add = TRUE
)
```



End of Problem 4

Problem 5: Beverage Sales Data

A grocery store sells 4 different brands of beverage:

- Kaff Blast, abbreviated as "KBL"
- Zow Kola, abbreviated as "ZKL"
- Mango Delight, abbreviated as "MNG"
- Kale and Tofu Juice, abbreviated as "KTJ"

The beverage brand and number of bottles sold are recorded for each individual transaction:

- The vector `problem.5.brand.data` is a character string vector that contains the brand name of the beverage sold.
- The vector `problem.5.number.of.bottles.sold.data` is a numeric vector that contains the number of bottles sold for.

Our goal in this problem is to visualize the relative proportions of the total revenues for the four different brands.

Part (a): Data inspection

Unfortunately, there is an immediate problem: the sales data has been inconsistently reported, and there are a number of different representations for each brand.

Thus, our first challenge is to clean the data so that each brand is represented by a single character string, and to get started we'll look at the first few values of the brand name vector.

To do this, directly display the first 8 values of `problem.8.brand.data`).

Solution

```
head( problem.5.brand.data, n = 8 )
```

```
## [1] "Kaff Blast" "Zow Kola"   "Zow"        "Kaff"       "MNG DEL"
## [6] "K BLAST"    "Kaff"       "K BLAST"
```

Part (b): Unique representatives

From part (a), it's clear that there are many different variants for representing the brands.

Let's get a complete listing of all the different variants by directly displaying a vector consisting of unique representatives for all the different values in `problem.5.brand.data`.

Solution

```
unique( problem.5.brand.data )
```

```
##  [1] "Kaff Blast"      "Zow Kola"        "Zow"             "Kaff"
##  [5] "MNG DEL"         "K BLAST"         "Z KOLA"          "Mango Delight"
##  [9] "Kale Tofu Juice" "Kale"            "KTJ"
```

Part (c): Repair the data

Convert all the different brand name variants in `problem.8.brand.data` to the standard abbreviations from the problem statement.

In other words, all the different representations for Kaff Blast should be converted to the abbreviation "KBL", and similarly for the other brands.

Save this repaired version of the brand name data in a variable, and directly display the first 8 values. (Hint: use a lookup vector.)

Also, directly display a vector of unique representatives for the values in the repaired version of the brand name data.

Solution

```
brand.name.lookup.vector <-
    c(
        "Kaff Blast" = "KBL",
        "Kaff" = "KBL",
        "K BLAST" = "KBL",
        "Zow Kola" = "ZKL",
        "Zow" = "ZKL",
        "Z KOLA" = "ZKL",
        "Mango Delight" = "MNG",
        "MNG DEL" = "MNG",
        "Kale Tofu Juice" = "KTJ",
        "Kale" = "KTJ",
        "KTJ" = "KTJ"
    )
```

```
repaired.brand.data <-
    brand.name.lookup.vector[
        problem.5.brand.data
    ]
```

```
head( repaired.brand.data, n = 8 )
```

```
## Kaff Blast   Zow Kola        Zow       Kaff    MNG DEL    K BLAST       Kaff
##      "KBL"      "ZKL"      "ZKL"      "KBL"      "MNG"      "KBL"      "KBL"
##    K BLAST
##      "KBL"
```

```
unique( repaired.brand.data )
```

```
## [1] "KBL" "ZKL" "MNG" "KTJ"
```

Part (d): Price per bottle

The price for a bottle for each beverage brand is:

Brand	Price
“KBL”	2.79
“ZKL”	2.99
“MNG”	3.99
“KTJ”	8.49

Construct a vector that consists of the price per bottle for the brand in each transaction, and save this vector in a variable.

Then directly display the first 8 values of this price vector.

Solution

```
price.lookup.vector <-
    c(
        "KBL" = 2.79,
        "ZKL" = 2.99,
        "MNG" = 3.99,
        "KTJ" = 8.49
    )
```

```
price.vector <-
    price.lookup.vector[
        repaired.brand.data
    ]

head( price.vector, n = 8 )
```

```
##  KBL  ZKL  ZKL  KBL  MNG  KBL  KBL  KBL
## 2.79 2.99 2.99 2.79 3.99 2.79 2.79 2.79
```

Part (e): Transaction sales amount

Using the vector of prices from part (d) and `problem.5.number.of.bottles.sold.data`, construct a vector consisting of the sales amount for each transaction, and save this vector in a variable.

Then directly display the first 8 values of this transaction sales amount vector.

Solution

```
transaction.amount.vector <-
    price.vector * problem.5.number.of.bottles.sold.data

head( transaction.amount.vector, n = 8 )
```

```
##   KBL   ZKL   ZKL   KBL   MNG   KBL   KBL   KBL
## 16.74 26.91 11.96 19.53  3.99  8.37 19.53 19.53
```

Part (f): Kaff Blast total sales amount

Select the values in the transaction sales amount vector from part (e) that correspond to a Kaff Blast transaction, and sum them to obtain the total sales amount for Kaff Blast.

Store this value in a vector, and report it using a `cat()` statement, formatting the value with 2 decimal places.

Solution

```
kaff.blast.transaction.amount.vector <-
    transaction.amount.vector[
        repaired.brand.data == "KBL"
    ]

kaff.blast.total.sales.amount <-
    sum( kaff.blast.transaction.amount.vector )

cat(
    "Kaff Blast total sales amount:",
    formatC(
        x = kaff.blast.total.sales.amount,
        format = "f",
        digits = 2,
        big.mark = ","
    ),
    "\n"
)
```

```
## Kaff Blast total sales amount: 6,933.15
```

Part (g): Zow Kola total sales amount

Select the values in the transaction sales amount vector from part (e) that correspond to a Zow Kola transaction, and sum them to obtain the total sales amount for Zow Kola.

Store this value in a vector, and report it using a `cat()` statement, formatting the value with 2 decimal places.

Solution

```
zow.kola.transaction.amount.vector <-
    transaction.amount.vector[
        repaired.brand.data == "ZKL"
    ]

zow.kola.total.sales.amount <-
    sum( zow.kola.transaction.amount.vector )

zow.kola.total.sales.amount
```

```
## [1] 3259.1
```

Part (h): Mango Delight total sales amount

Select the values in the transaction sales amount vector from part (e) that correspond to a Mango Delight transaction, and sum them to obtain the total sales amount for Mango Delight.

Store this value in a vector, and report it using a `cat()` statement, formatting the value with 2 decimal places.

Solution

```
mango.delight.transaction.amount.vector <-
    transaction.amount.vector[
        repaired.brand.data == "MNG"
    ]

mango.delight.total.sales.amount <-
    sum( mango.delight.transaction.amount.vector )

mango.delight.total.sales.amount
```

```
## [1] 917.7
```

Part (i): Kale and Tofu Juice total sales amount

Select the values in the transaction sales amount vector from part (e) that correspond to a Kale and Tofu Juice transaction, and sum them to obtain the total sales amount for Kale and Tofu Juice.

Store this value in a vector, and report it using a `cat()` statement, formatting the value with 2 decimal places.

Solution

```
kale.tofu.juice.transaction.amount.vector <-
    transaction.amount.vector[
        repaired.brand.data == "KTJ"
    ]

kale.tofu.juice.total.sales.amount <-
    sum( kale.tofu.juice.transaction.amount.vector )

kale.tofu.juice.total.sales.amount
```

```
## [1] 755.61
```

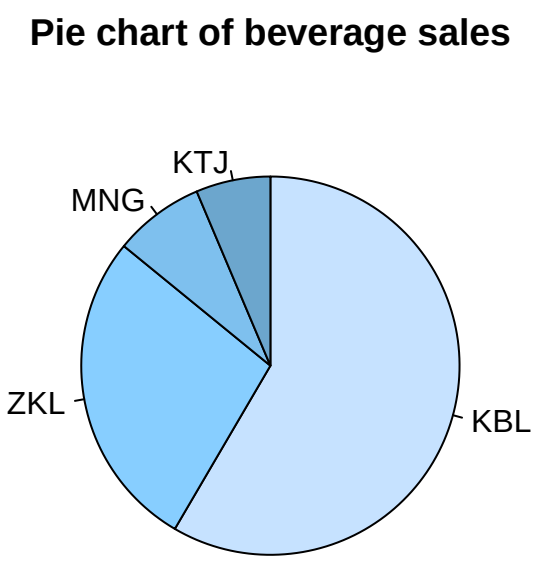
Part (j): Pie chart

Using the total sales amounts for each brand from parts (f) through (i), construct a pie chart to visualize the relative proportions of each brand for the total sales revenue.

- Use the labels "KBL", "ZKL", "MNG", and "KTJ" for the pie slices.
- Set the rotation to clockwise.
- Include a main title.
- Explicitly specify colors for each pie slice.

Solution

```
total.sales.vector <-
    c(
        kaff.blast.total.sales.amount,
        zow.kola.total.sales.amount,
        mango.delight.total.sales.amount,
        kale.tofu.juice.total.sales.amount
    )

pie(
    x = total.sales.vector,
    labels = c( "KBL", "ZKL", "MNG", "KTJ" ),
    main = "Pie chart of beverage sales",
    clockwise = TRUE,
    col = c( "slategray1", "skyblue1", "skyblue2", "skyblue3")
)
```



End of Problem 5

Problem 6: Newton's Method

In Problem Set 4, Problem 1, we studied Newton's method, in which we make an initial estimate for the root of a function, and then use a special formula to calculate an improved estimate. In that problem, we focused on quadratic polynomials, which are polynomials where the highest power is 2.

In this problem, we'll learn how to Newton's method for cubic polynomials, which are polynomials where the highest power is 3.

The standard form of a cubic polynomial is:

$$f(x) \;=\; ax^3 + bx^2 + cx + d$$

The special formula that we use to iteratively improve the root estimate for a cubic polynomial is:

$$x_{i+1} \;=\; \frac{2ax_i^3 + bx_i^2 - d}{3ax_i^2 + 2bx_i + c}$$

In this problem, we'll use Newton's method to determine the roots of this cubic polynomial:

$$f(x) = x^3 - 3x^2 - 45x + 175$$

Part (a): First root, first iteration

Let's start by estimating the first root by $x_1 = -4$.

Then use the special formula for Newton's method for a cubic polynomial to obtain the second estimate. Report your result by using a `cat()` statement, formatting the value with 5 decimal places.

Solution

```
a <- 1

b <- -3

c <- -45

d <- 175
```

```
first.estimate <- -4
```

```
second.estimate <-
    (2 * a * first.estimate^3 + b * first.estimate^2 - d) /
    (3 * a * first.estimate^2 + 2 * b * first.estimate + c)
```

```
cat(
    "Second estimate:",
    formatC(
        second.estimate,
        format = "f",
        digits = 5
    ),
    "\n"
)
```

```
## Second estimate: -13.00000
```

Part (b): First root, third estimate

Using the second estimate that you calculated in part (a), use Newton's method for a cubic polynomial to calculate a third estimate for this root. Report your result by using a `cat()` statement, displaying the value with 5 decimal places.

Solution

```
third.estimate <-
    (2 * a * second.estimate^3 + b * second.estimate^2 - d) /
    (3 * a * second.estimate^2 + 2 * b * second.estimate + c)
```

```
cat(
    "Third estimate:",
    formatC(
        third.estimate,
        format = "f",
        digits = 5
    ),
    "\n"
)
```

```
## Third estimate: -9.40000
```

Part (c): Iteration

Construct a `for()` loop that will start with the initial estimate of $x_1 = -4$ and then iteratively calculates the successive approximations to the root by using Newton's method for a cubic polynomial. For each iteration of the `for()` loop, print out the value of the estimate after applying the special formula, so that the first iteration will print out the value of the second estimate, the second iteration will print out the value of the third estimate, and so on. Display the values with a `cat()` statement, formatting the values with 5 decimal places.

You'll have to decide how many iterations are needed. The `cat()` statements are printing out the value of the estimate to 5 decimal places, so run the `for()` loop a sufficient number of times so that this estimate doesn't change any more.

Finally, when you've determined a good approximation to this root, store it in a variable.

When you print out the values you should annotate them so that we know what it is that you are reporting; don't just print out the numbers without any explanation.

IMPORTANT: You MUST use a `for()` loop for this problem – do not create a bunch of dedicated variables and then manually write out all the code to update them.

Solution

```
current.estimate <- -4

for( current.iteration in 1:8 ) {

    current.estimate <-
        (2 * a * current.estimate^3 + b * current.estimate^2 - d) /
        (3 * a * current.estimate^2 + 2 * b * current.estimate + c)
```

```
    cat(
        current.iteration,
        ". Current estimate:",
        formatC(
            current.estimate,
            format = "f",
            digits = 5
        ),
        "\n",
        sep = ""
    )
}
```

```
## 1. Current estimate:-13.00000
## 2. Current estimate:-9.40000
## 3. Current estimate:-7.60000
## 4. Current estimate:-7.05217
## 5. Current estimate:-7.00045
## 6. Current estimate:-7.00000
## 7. Current estimate:-7.00000
## 8. Current estimate:-7.00000
```

```
first.root <- current.estimate
```

Part (d): Second root

Now use Newton's method to determine the second root of the cubic polynomial. Start with an initial estimate of $x_1 = 10$, and use a `for()` loop to calculate the successive approximations using the special formual for Newton's method for cubic polynomials. For each iteration of the `for()` loop, print out the value of the estimate after applying the special formula, so that the first iteration will print out the value of the second estimate, the second iteration will print out the value of the third estimate, and so on. Display the values with a `cat()` statement, formatting the values with 5 decimal places.

You'll have to decide how many iterations are needed. The `cat()` statements are printing out the value of the estimate to 5 decimal places, so run the `for()` loop a sufficient number of times so that this estimate doesn't change any more.

Finally, when you've determined a good approximation to this root, store it in a variable.

When you print out the values you should annotate them so that we know what it is that you are reporting; don't just print out the numbers without any explanation.

IMPORTANT: You MUST use a `for()` loop for this problem – do not create a bunch of dedicated variables and then manually write out all the code to update them.

Solution

```
current.estimate <- 10

for( current.iteration in 1:25 ) {

    current.estimate <-
        (2 * a * current.estimate^3 + b * current.estimate^2 - d) /
        (3 * a * current.estimate^2 + 2 * b * current.estimate + c)
```

```
    cat(
        current.iteration,
        ". Current estimate:",
        formatC(
            current.estimate,
            format = "f",
            digits = 5
        ),
        "\n",
        sep = ""
    )
}
```

```
## 1. Current estimate:7.82051
## 2. Current estimate:6.53279
## 3. Current estimate:5.80747
## 4. Current estimate:5.41607
## 5. Current estimate:5.21147
## 6. Current estimate:5.10664
## 7. Current estimate:5.05355
## 8. Current estimate:5.02684
## 9. Current estimate:5.01343
## 10. Current estimate:5.00672
## 11. Current estimate:5.00336
## 12. Current estimate:5.00168
## 13. Current estimate:5.00084
## 14. Current estimate:5.00042
## 15. Current estimate:5.00021
## 16. Current estimate:5.00011
## 17. Current estimate:5.00005
## 18. Current estimate:5.00003
## 19. Current estimate:5.00001
## 20. Current estimate:5.00001
## 21. Current estimate:5.00000
## 22. Current estimate:5.00000
## 23. Current estimate:5.00000
## 24. Current estimate:5.00000
## 25. Current estimate:5.00000
```

```
second.root <- current.estimate
```

Part (e): Graph

Now let's check our work by constructing a graph of this cubic polynomial.

Use the `curve()` function to plot the graph of this cubic polynomial:

- The x-axis should range from -10 to +10.
- The y-axis should range from -200 to 300.
- Specify a main title, and titles for the x- and y-axes.

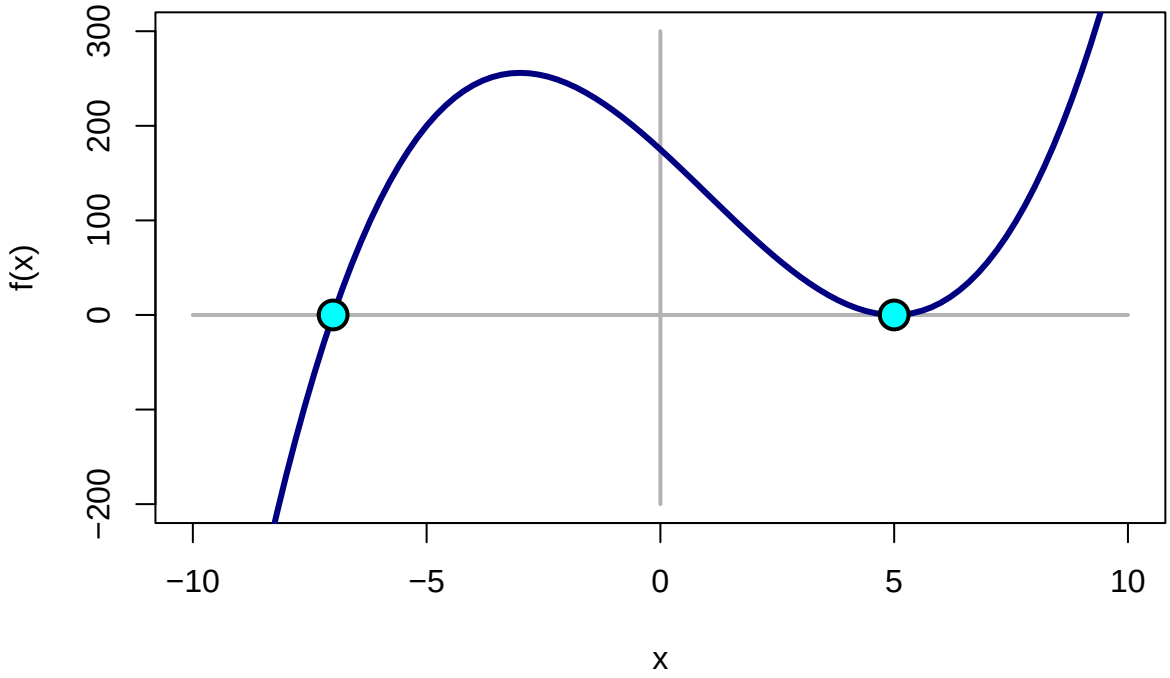
- Explicitly specify the line type, line width, and color.
- Include horizontal and vertical reference lines.

Once you've drawn the curve, plot one point at the location of the first root that you calculated in part (c) and another point at the location of the second root that you calculated in part (d). Explicitly specify the size, shape, and color of the points, and annotate them with text.

Solution

```
plot(
    x = NULL,
    xlim = c( -10, 10 ),
    ylim = c( -200, +300 ),
    xlab = "x",
    ylab = "f(x)"
)

segments(
    x0 = -10,
    y0 = 0,
    x1 = +10,
    y1 = 0,
    lty = "solid",
    lwd = 2,
    col = "gray70"
)

segments(
    x0 = 0,
    y0 = -200,
    x1 = 0,
    y1 = +300,
    lty = "solid",
    lwd = 2,
    col = "gray70"
)


curve(
    expr = x^3 - 3*x^2 - 45*x + 175,
    lty = "solid",
    lwd = 3,
    col = "navy",
    add = TRUE
)

points(
    x = first.root,
    y = 0,
    pch = 21,
    lwd = 2,
    cex = 2,
    col = "black",
    bg = "cyan"
```

```
)

points(
    x = second.root,
    y = 0,
    pch = 21,
    lwd = 2,
    cex = 2,
    col = "black",
    bg = "cyan"
)
```



End of Problem 6

Problem 7: Quarterback Rating

In the National Football League, a quarterback is assigned a "passer rating" based on 5 performance statistics:

- The number of passing attempts, denoted ATT.
- The number of completions, denoted COMP.
- The total number of passing yards, denoted TOT.
- The number of touchdown passes, denoted TD.
- The number of interceptions, denoted INT.

To calculate the passer rating, we first have to compute four coefficients, denoted a, b, c, and d.

The formula for a is:

$$a = \left(\frac{\text{COMP}}{\text{ATT}} - 0.3\right) \times 5$$

The formula for b is:

$$b = \left(\frac{\text{TOT}}{\text{ATT}} - 3\right) \times 0.25$$

The formula for c is:

$$c = \left(\frac{\text{TD}}{\text{ATT}}\right) \times 20$$

The formula for d is:

$$d = 2.375 - \left(\frac{\text{INT}}{\text{ATT}} \times 25\right)$$

For all of these calculations:

- The maximum value for any coefficient is 2.375, so if a calculation leads to a value greater than this then the coefficient is assigned the value 2.375.
- The minimum value for any coefficient is 0, so if a calculation leads to a negative number then the coefficient is assigned the value 0.

The passer rating R is then calculated as:

$$R = \frac{a + b + c + d}{6} \times 100$$

The data for this problem is contained in 5 vectors:

- The vector `problem.7.completions.data` contains the data for the number of completions.
- The vector `problem.7.attempts.data` contains the data for the number of attempts.
- The vector `problem.7.total.yards.data` contains the data for the total number of passing yards.
- The vector `problem.7.touchdown.pass.data` contains the data for the total number of touchdown passes.
- The vector `problem.7.interception.data` contains the data for the total number of interceptions.

Part (a): Coefficient a

To calculate the coefficient a, we have:

$$a = \left(\frac{\text{COMP}}{\text{ATT}} - 0.3 \right) \times 5$$

- If the result of this calculation is greater than 2.375, then the value 2.375 is assigned to a.
- If the result of this calculation is less than 0, then the value 0 is assigned to a.

Using the vectors `problem.7.completions.data` and `problem.7.attempts.data`, construct a vector of the a coefficients, and save this vector in a variable.

Then display the first 8 values of this vector by using a `cat()` statement, formatting the values with 2 decimal places.

Solution

```
coefficient.a.vector <-
    (problem.7.completions.data / problem.7.attempts.data - 0.3 ) * 5

coefficient.a.vector <-
    ifelse(
        coefficient.a.vector < 0,
        0,
        coefficient.a.vector
    )

coefficient.a.vector <-
    ifelse(
        coefficient.a.vector > 2.375,
        2.375,
        coefficient.a.vector
    )



cat(
    "a coefficients:",
    formatC(
        x = head( coefficient.a.vector ),
        format = "f",
        digits = 2
    )
)
```

```
## a coefficients: 0.00 0.00 0.01 0.00 0.00 1.48
```

Part (b): Coefficient b

To calculate the coefficient b, we have:

$$b = \left(\frac{\text{TOT}}{\text{ATT}} - 3 \right) \times 0.25$$

- If the result of this calculation is greater than 2.375, then the value 2.375 is assigned to a.
- If the result of this calculation is less than 0, then the value 0 is assigned to a.

Using the vectors `problem.7.total.yards.data` and `problem.7.attempts.data`, construct a vector of the b coefficients, and save this vector in a variable.

Then display the first 8 values of this vector by using a `cat()` statement, formatting the values with 2 decimal places.

Solution

```
coefficient.b.vector <-
    (problem.7.total.yards.data / problem.7.attempts.data - 3 ) * 0.25

coefficient.b.vector <-
    ifelse(
        coefficient.b.vector < 0,
        0,
        coefficient.b.vector
    )

coefficient.b.vector <-
    ifelse(
        coefficient.b.vector > 2.375,
        2.375,
        coefficient.b.vector
    )



cat(
    "b coefficients:",
    formatC(
        x = head( coefficient.b.vector ),
        format = "f",
        digits = 2
    )
)
```

```
## b coefficients: 0.00 0.00 0.17 0.00 0.00 0.77
```

Part (c): Coefficient c

To calculate the coefficient c, we have:

$$c = \left(\frac{\text{TD}}{\text{ATT}} \right) \times 20$$

- If the result of this calculation is greater than 2.375, then the value 2.375 is assigned to a.
- If the result of this calculation is less than 0, then the value 0 is assigned to a.

Using the vectors `problem.7.touchdown.pass.data` and `problem.7.attempts.data`, construct a vector of the c coefficients, and save this vector in a variable.

Then display the first 8 values of this vector by using a `cat()` statement, formatting the values with 2 decimal places.

Solution

```
coefficient.c.vector <-
    (problem.7.touchdown.pass.data / problem.7.attempts.data ) * 20

coefficient.c.vector <-
    ifelse(
        coefficient.c.vector < 0,
        0,
        coefficient.c.vector
    )

coefficient.c.vector <-
    ifelse(
        coefficient.c.vector > 2.375,
        2.375,
        coefficient.c.vector
    )



cat(
    "c coefficients:",
    formatC(
        x = head( coefficient.c.vector ),
        format = "f",
        digits = 2
    )
)
```

```
## c coefficients: 0.95 2.07 0.53 1.02 0.45 0.28
```

Part (d): Coefficient d

To calculate the coefficient d, we have:

$$d \;=\; 2.375 - \left(\frac{\text{INT}}{\text{ATT}} \times 25 \right)$$

- If the result of this calculation is greater than 2.375, then the value 2.375 is assigned to a.
- If the result of this calculation is less than 0, then the value 0 is assigned to a.

Using the vectors `problem.7.interception.data` and `problem.7.attempts.data`, construct a vector of the d coefficients, and save this vector in a variable.

Then display the first 8 values of this vector by using a `cat()` statement, formatting the values with 2 decimal places.

Solution

```
coefficient.d.vector <-
    2.375 - (problem.7.interception.data / problem.7.attempts.data * 25 )

coefficient.d.vector <-
    ifelse(
        coefficient.d.vector < 0,
        0,
        coefficient.d.vector
    )

coefficient.d.vector <-
    ifelse(
        coefficient.d.vector > 2.375,
        2.375,
        coefficient.d.vector
    )



cat(
    "d coefficients:",
    formatC(
        x = head( coefficient.d.vector ),
        format = "f",
        digits = 2
    )
)
```

```
## d coefficients: 1.93 2.23 1.62 1.57 2.26 0.36
```

Part (e): Passer rating

The passer rating R is then calculated as:

$$R = \frac{a+b+c+d}{6} \times 100$$

Use the vectors that you constructed in parts (a) through (d) to construct a vector of passer ratings, and save this in a variable.

The display the first 8 elements of this vector by using a `cat()` statement, formatting the values with 2 decimal places.

Solution

```
passer.rating.vector <-
    (coefficient.a.vector +
         coefficient.b.vector +
         coefficient.c.vector +
         coefficient.d.vector ) / 6 * 100

cat(
    "Passer ratings:",
    formatC(
```

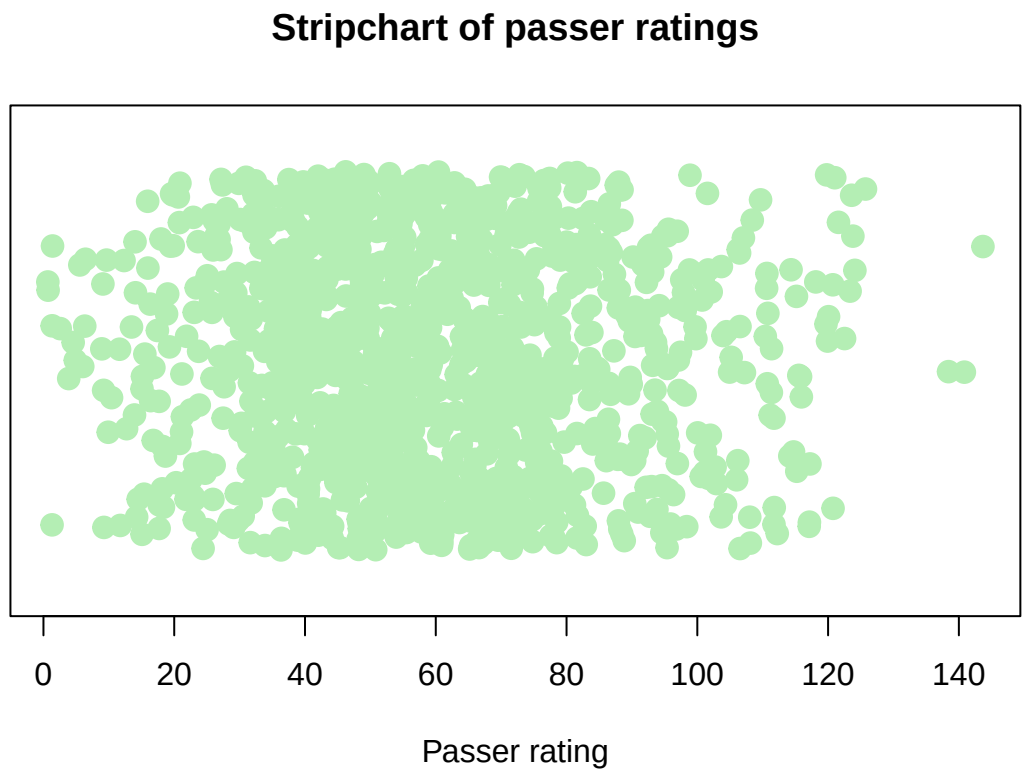
```
        x = head( passer.rating.vector, n = 8),
        format = "f",
        digits = 2
    ),
    "\n"
)
```

```
## Passer ratings: 48.00 71.76 38.58 43.21 45.18 48.21 65.80 97.62
```

Part (f): Stripchart

Solution

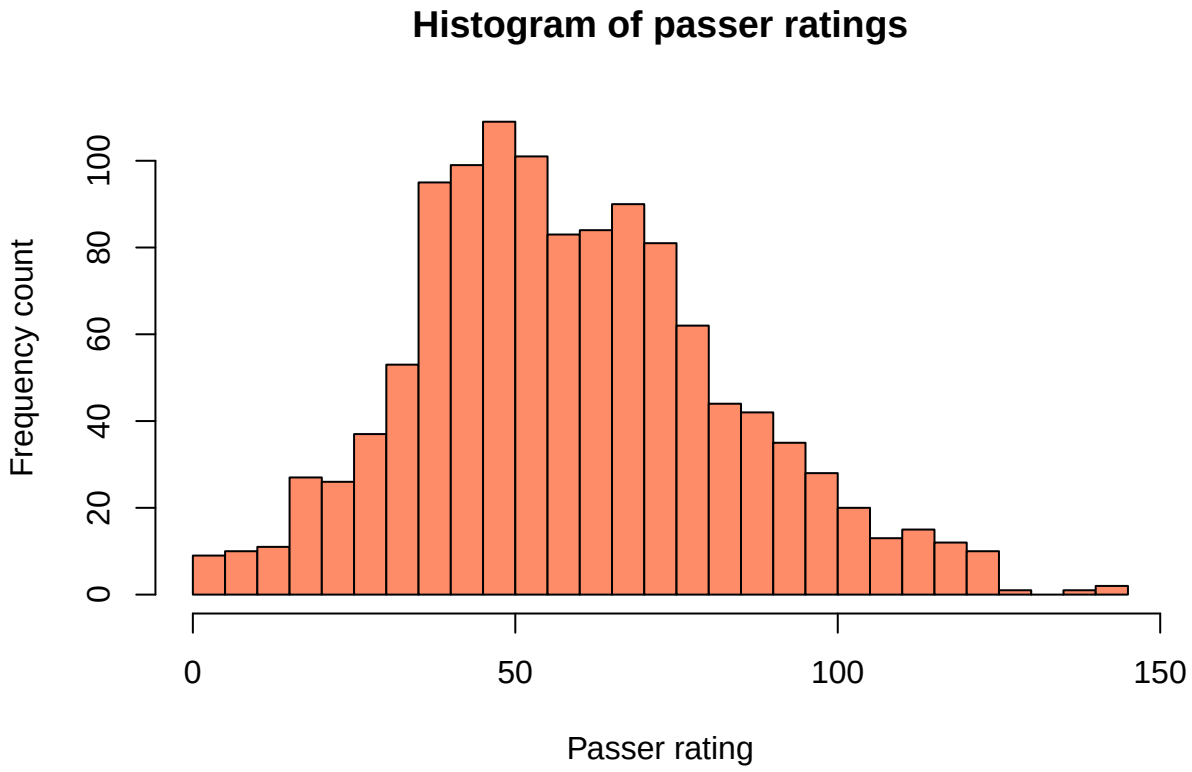
```
stripchart(
    x = passer.rating.vector,
    ylim = c(0, 2),
    main = "Stripchart of passer ratings",
    xlab = "Passer rating",
    method = "jitter",
    jitter = 0.8,
    pch = 19,
    cex = 1.5,
    col = "darkseagreen2"
)
```



Part (g): Histogram

Solution

```
hist(
    x = passer.rating.vector,
    main = "Histogram of passer ratings",
    xlab = "Passer rating",
    ylab = "Frequency count",
    breaks = 50,
    col = "salmon1"
)
```

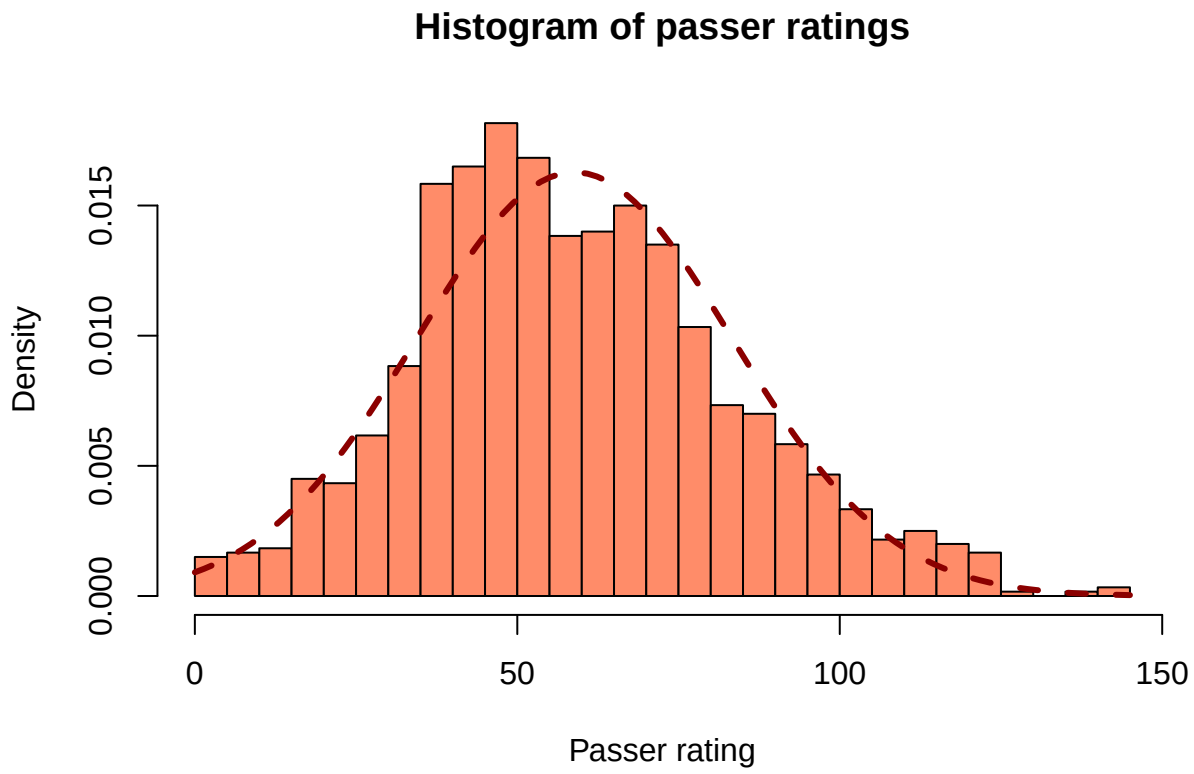


Part (h): Histogram with normal density curve

Solution

```
hist(
    x = passer.rating.vector,
    main = "Histogram of passer ratings",
    xlab = "Passer rating",
    ylab = "Density",
    prob = TRUE,
    breaks = 50,
```

```
    col = "salmon1"
)

sample.mean <-
    mean( passer.rating.vector )

sample.sd <-
    sd( passer.rating.vector )

curve(
    expr = dnorm(x, mean = sample.mean, sd = sample.sd ),
    lty = "dashed",
    lwd = 3,
    col = "darkred",
    add = TRUE
)
```



End of Problem 7

Problem 8: Mandala

In this problem, we'll construct a mystical mandala and encrust it with precious jewels!

We'll start by constructing a basic plotting display, and then successively developing each component of the picture.

Thus, it will be to your advantage to write your code so that it's easy to copy it and modify it.

Part (a): Plotting region

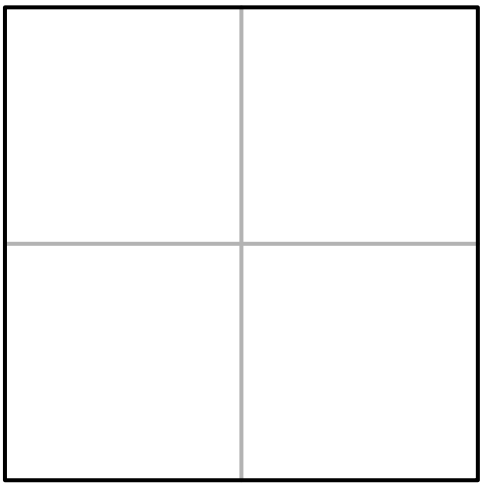
We'll start by constructing a basic plotting display:

- Use a graphical parameter to set the plot type to a square display.
- Construct a completely blank plotting region without axes, titles, or a surrounding box:
  - The x-axis should range from -1 to +1.
  - The y-axis should range from -1 to +1.
- Draw a horizontal line segment from the point (-1, 0) to the point (+1, 0).
  - The line type should be "solid".
  - The line width should be 2.
  - The line color should be "grey70".
- Draw a vertical line segment from the point (0, -1) to the point (0, +1).
  - The line type should be "solid".
  - The line width should be 2.
  - The line color should be "grey70".
- Finally, draw a connected sequence of lines segments with the endpoints (-1, -1), (-1, +1), (+1, +1), and (+1, -1).
  - The line type should be "solid".
  - The line width should be 3.
  - The line color should be "black".

Solution

```
par( pty = "s" )

plot(
    x = NULL,
    xlim = c( -1, +1 ),
    ylim = c( -1, +1 ),
    xlab = "",
    ylab = "",
    axes = FALSE
)

segments(
```

```
    x0 = 0,
    y0 = -1,
    x1 = 0,
    y1 = +1,
    lty = "solid",
    lwd = 2,
    col = "gray70"
)



segments(
    x0 = -1,
    y0 = 0,
    x1 = +1,
    y1 = 0,
    lty = "solid",
    lwd = 2,
    col = "gray70"
)


polygon(
    x = c( 1, -1, -1, 1 ),
    y = c( 1, 1, -1, -1 ),
    lty = "solid",
    lwd = 2,
    border = "black"
)
```



Part (b): Outer circle

Now let's draw the first figure of the mandala: a large circle.

We'll use three vectors for this problem:

- The vector `problem.8.category.data` is a character string vector consisting of the names of various components of the graph.
- The vector `problem.8.x.data` is a numeric vector consisting of the x-coordinates of points.
- The vector `problem.8.y.data` is a numeric vector consisting of the y-coordinates of points.

The strategy will be to use the categories in `problem.8.category.data` to select x-values from `problem.8.x.data` and y-values from `problem.8.y.data`:

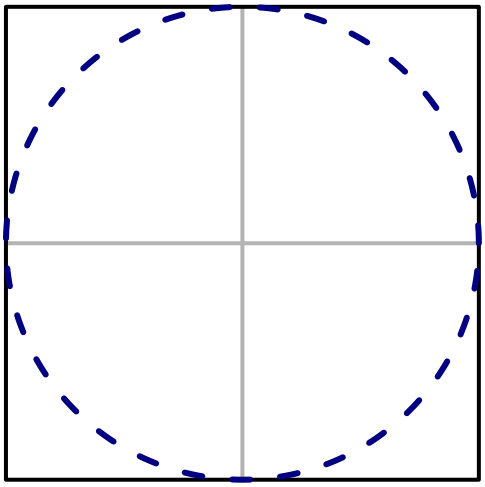
- First, copy your code from part (a).
- Select the x-values in `problem.8.x.data` that correspond to the elements in `problem.8.category.data` that are equal to "Outer Circle".
- Select the y-values in `problem.8.y.data` that correspond to the elements in `problem.8.category.data` that are equal to "Outer Circle".
- Then use the `lines()` function to draw a connected sequence of line segments through the points in the filtered x- and y-values:

- The line type should be “dashed”.
- The line width should be 3.
- The line color should be “navy”.

Solution

```
par( pty = "s" )

plot(
    x = NULL,
    xlim = c( -1, +1 ),
    ylim = c( -1, +1 ),
    xlab = "",
    ylab = "",
    axes = FALSE
)

segments(
    x0 = 0,
    y0 = -1,
    x1 = 0,
    y1 = +1,
    lty = "solid",
    lwd = 2,
    col = "gray70"
)



segments(
    x0 = -1,
    y0 = 0,
    x1 = +1,
    y1 = 0,
    lty = "solid",
    lwd = 2,
    col = "gray70"
)


polygon(
    x = c( 1, -1, -1, 1 ),
    y = c( 1, 1, -1, -1 ),
    lty = "solid",
    lwd = 2,
    border = "black"
)

outer.circle.x.vector <-
    problem.8.x.data[ problem.8.category.data == "Outer Circle" ]

outer.circle.y.vector <-
```

```
    problem.8.y.data[ problem.8.category.data == "Outer Circle" ]

lines(
    x = outer.circle.x.vector,
    y = outer.circle.y.vector,
    lty = "dashed",
    lwd = 3,
    col = "navy"
)
```



Part (c): Inner circle

Now let's draw the second figure of the mandala: a smaller inner circle.

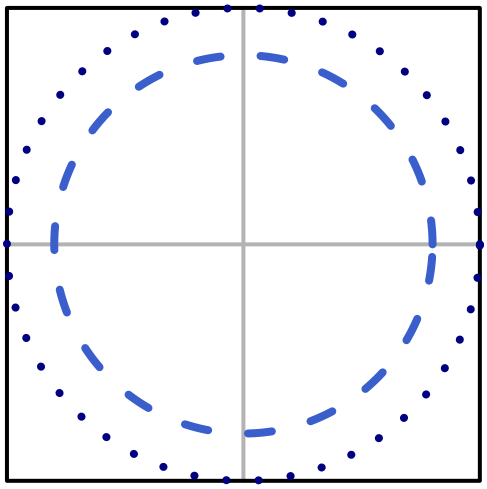
- First, copy your code from part (b).
- Select the x-values in `problem.8.x.data` that correspond to the elements in `problem.8.category.data` that are equal to "Inner Circle".
- Select the y-values in `problem.8.y.data` that correspond to the elements in `problem.8.category.data` that are equal to "Inner Circle".
- Then use the `lines()` function to draw a connected sequence of line segments through the points in the filtered x- and y-values:
  - The line type should be "dotted".

- The line width should be 2.
- The line color should be “royalblue3”.

Solution

```
par( pty = "s" )

plot(
    x = NULL,
    xlim = c( -1, +1 ),
    ylim = c( -1, +1 ),
    xlab = "",
    ylab = "",
    axes = FALSE
)

segments(
    x0 = 0,
    y0 = -1,
    x1 = 0,
    y1 = +1,
    lty = "solid",
    lwd = 2,
    col = "gray70"
)



segments(
    x0 = -1,
    y0 = 0,
    x1 = +1,
    y1 = 0,
    lty = "solid",
    lwd = 2,
    col = "gray70"
)


polygon(
    x = c( 1, -1, -1, 1 ),
    y = c( 1, 1, -1, -1 ),
    lty = "solid",
    lwd = 2,
    border = "black"
)

outer.circle.x.vector <-
    problem.8.x.data[ problem.8.category.data == "Outer Circle" ]

outer.circle.y.vector <-
    problem.8.y.data[ problem.8.category.data == "Outer Circle" ]
```

```
lines(
    x = outer.circle.x.vector,
    y = outer.circle.y.vector,
    lty = "dotted",
    lwd = 4,
    col = "navy"
)


inner.circle.x.vector <-
    problem.8.x.data[ problem.8.category.data == "Inner Circle" ]

inner.circle.y.vector <-
    problem.8.y.data[ problem.8.category.data == "Inner Circle" ]

lines(
    x = inner.circle.x.vector,
    y = inner.circle.y.vector,
    lty = "dashed",
    lwd = 4,
    col = "royalblue3"
)
```



Part (d): First triangle

Now we'll draw the third figure of the mandala: an inscribed triangle.

- First, copy your code from part (c).
- Select the x-values in `problem.8.x.data` that correspond to the elements in `problem.8.category.data` that are equal to "Triangle 1".
- Select the y-values in `problem.8.y.data` that correspond to the elements in `problem.8.category.data` that are equal to "Triangle 1".
- Then use the `polygon()` function to draw a connected sequence of line segments through the points in the filtered x- and y-values:
  - The line type should be "solid".
  - The line width should be 2.
  - The line color should be "cadetblue3".

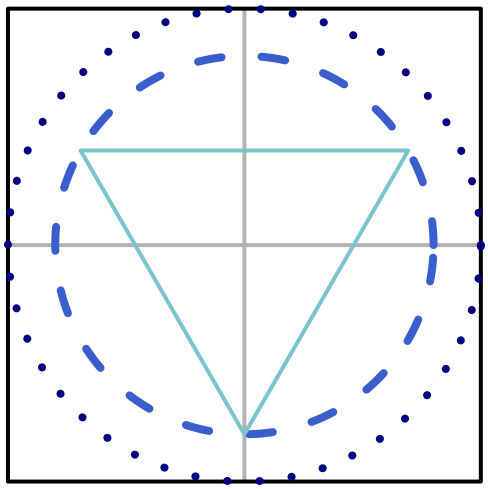
The polygon should not have a fill color, but instead should just have a black border. (The preceding sentence contains a subtle hint about the correct optional argument to use.) As a check, the light-grey lines that cross in the center of the picture should still be visible.

Solution

```
par( pty = "s" )

plot(
    x = NULL,
    xlim = c( -1, +1 ),
    ylim = c( -1, +1 ),
    xlab = "",
    ylab = "",
    axes = FALSE
)

segments(
    x0 = 0,
    y0 = -1,
    x1 = 0,
    y1 = +1,
    lty = "solid",
    lwd = 2,
    col = "gray70"
)



segments(
    x0 = -1,
    y0 = 0,
    x1 = +1,
    y1 = 0,
    lty = "solid",
```

```
    lwd = 2,
    col = "gray70"
)


polygon(
    x = c( 1, -1, -1, 1 ),
    y = c( 1, 1, -1, -1 ),
    lty = "solid",
    lwd = 2,
    border = "black"
)

outer.circle.x.vector <-
    problem.8.x.data[ problem.8.category.data == "Outer Circle" ]

outer.circle.y.vector <-
    problem.8.y.data[ problem.8.category.data == "Outer Circle" ]

lines(
    x = outer.circle.x.vector,
    y = outer.circle.y.vector,
    lty = "dotted",
    lwd = 4,
    col = "navy"
)


inner.circle.x.vector <-
    problem.8.x.data[ problem.8.category.data == "Inner Circle" ]

inner.circle.y.vector <-
    problem.8.y.data[ problem.8.category.data == "Inner Circle" ]

lines(
    x = inner.circle.x.vector,
    y = inner.circle.y.vector,
    lty = "dashed",
    lwd = 4,
    col = "royalblue3"
)


triangle.1.x.vector <-
    problem.8.x.data[ problem.8.category.data == "Triangle 1" ]

triangle.1.y.vector <-
    problem.8.y.data[ problem.8.category.data == "Triangle 1" ]

polygon(
    x = triangle.1.x.vector,
    y = triangle.1.y.vector,
    lty = "solid",
```

```
    lwd = 2,
    border = "cadetblue3"
)
```



Part (e): Second triangle

Now we'll draw the fourth figure of the mandala: another inscribed triangle.

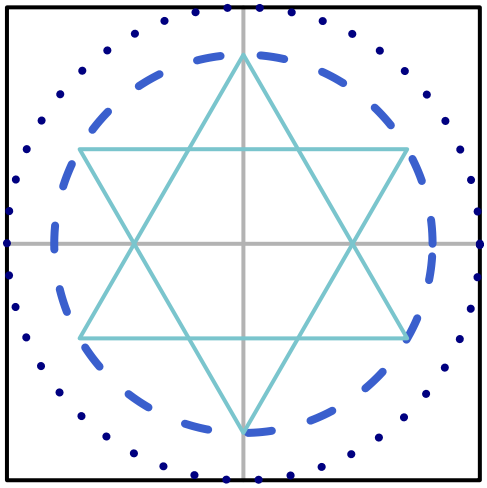
- First, copy your code from part (d).
- Select the x-values in `problem.8.x.data` that correspond to the elements in `problem.8.category.data` that are equal to "Triangle 2".
- Select the y-values in `problem.8.y.data` that correspond to the elements in `problem.8.category.data` that are equal to "Triangle 2".
- Then use the `polygon()` function to draw a connected sequence of line segments through the points in the filtered x- and y-values:
  - The line type should be "solid".
  - The line width should be 2.
  - The line color should be "cadetblue3".

The polygon should not have a fill color, but instead should just have a black border. (The preceding sentence contains a subtle hint about the correct optional argument to use.) As a check, the light-grey lines that cross in the center of the picture should still be visible.

Solution

```
par( pty = "s" )

plot(
    x = NULL,
    xlim = c( -1, +1 ),
    ylim = c( -1, +1 ),
    xlab = "",
    ylab = "",
    axes = FALSE
)

segments(
    x0 = 0,
    y0 = -1,
    x1 = 0,
    y1 = +1,
    lty = "solid",
    lwd = 2,
    col = "gray70"
)



segments(
    x0 = -1,
    y0 = 0,
    x1 = +1,
    y1 = 0,
    lty = "solid",
    lwd = 2,
    col = "gray70"
)


polygon(
    x = c( 1, -1, -1, 1 ),
    y = c( 1, 1, -1, -1 ),
    lty = "solid",
    lwd = 2,
    border = "black"
)

outer.circle.x.vector <-
    problem.8.x.data[ problem.8.category.data == "Outer Circle" ]

outer.circle.y.vector <-
    problem.8.y.data[ problem.8.category.data == "Outer Circle" ]
```

```
lines(
    x = outer.circle.x.vector,
    y = outer.circle.y.vector,
    lty = "dotted",
    lwd = 4,
    col = "navy"
)


inner.circle.x.vector <-
    problem.8.x.data[ problem.8.category.data == "Inner Circle" ]

inner.circle.y.vector <-
    problem.8.y.data[ problem.8.category.data == "Inner Circle" ]

lines(
    x = inner.circle.x.vector,
    y = inner.circle.y.vector,
    lty = "dashed",
    lwd = 4,
    col = "royalblue3"
)


triangle.1.x.vector <-
    problem.8.x.data[ problem.8.category.data == "Triangle 1" ]

triangle.1.y.vector <-
    problem.8.y.data[ problem.8.category.data == "Triangle 1" ]

polygon(
    x = triangle.1.x.vector,
    y = triangle.1.y.vector,
    lty = "solid",
    lwd = 2,
    border = "cadetblue3"
)

triangle.2.x.vector <-
    problem.8.x.data[ problem.8.category.data == "Triangle 2" ]

triangle.2.y.vector <-
    problem.8.y.data[ problem.8.category.data == "Triangle 2" ]

polygon(
    x = triangle.2.x.vector,
    y = triangle.2.y.vector,
    lty = "solid",
    lwd = 2,
    border = "cadetblue3"
)
```



Part (f): Hexagon triangle

Now we'll draw the fifth figure of the mandala: an inscribed hexagon.

- First, copy your code from part (e).
- Select the x-values in `problem.8.x.data` that correspond to the elements in `problem.8.category.data` that are equal to "Hexagon".
- Select the y-values in `problem.8.y.data` that correspond to the elements in `problem.8.category.data` that are equal to "Hexagon".
- Then use the `polygon()` function to draw a connected sequence of line segments through the points in the filtered x- and y-values:
  - The line type should be "solid".
  - The line width should be 2.
  - The line color should be "cadetblue3".

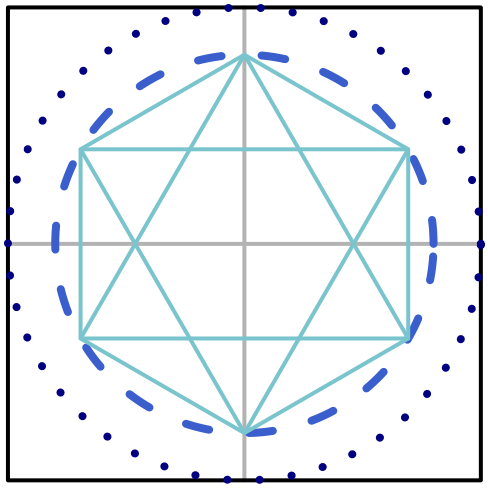
The polygon should not have a fill color, but instead should just have a black border. (The preceding sentence contains a subtle hint about the correct optional argument to use.) As a check, the light-grey lines that cross in the center of the picture should still be visible.

Solution

```
par( pty = "s" )

plot(
    x = NULL,
    xlim = c( -1, +1 ),
    ylim = c( -1, +1 ),
    xlab = "",
    ylab = "",
    axes = FALSE
)

segments(
    x0 = 0,
    y0 = -1,
    x1 = 0,
    y1 = +1,
    lty = "solid",
    lwd = 2,
    col = "gray70"
)



segments(
    x0 = -1,
    y0 = 0,
    x1 = +1,
    y1 = 0,
    lty = "solid",
    lwd = 2,
    col = "gray70"
)


polygon(
    x = c( 1, -1, -1, 1 ),
    y = c( 1, 1, -1, -1 ),
    lty = "solid",
    lwd = 2,
    border = "black"
)

outer.circle.x.vector <-
    problem.8.x.data[ problem.8.category.data == "Outer Circle" ]

outer.circle.y.vector <-
    problem.8.y.data[ problem.8.category.data == "Outer Circle" ]

lines(
    x = outer.circle.x.vector,
    y = outer.circle.y.vector,
    lty = "dotted",
    lwd = 4,
```

```
    col = "navy"
)


inner.circle.x.vector <-
    problem.8.x.data[ problem.8.category.data == "Inner Circle" ]

inner.circle.y.vector <-
    problem.8.y.data[ problem.8.category.data == "Inner Circle" ]

lines(
    x = inner.circle.x.vector,
    y = inner.circle.y.vector,
    lty = "dashed",
    lwd = 4,
    col = "royalblue3"
)


triangle.1.x.vector <-
    problem.8.x.data[ problem.8.category.data == "Triangle 1" ]

triangle.1.y.vector <-
    problem.8.y.data[ problem.8.category.data == "Triangle 1" ]

polygon(
    x = triangle.1.x.vector,
    y = triangle.1.y.vector,
    lty = "solid",
    lwd = 2,
    border = "cadetblue3"
)

triangle.2.x.vector <-
    problem.8.x.data[ problem.8.category.data == "Triangle 2" ]

triangle.2.y.vector <-
    problem.8.y.data[ problem.8.category.data == "Triangle 2" ]

polygon(
    x = triangle.2.x.vector,
    y = triangle.2.y.vector,
    lty = "solid",
    lwd = 2,
    border = "cadetblue3"
)


hexagon.x.vector <-
    problem.8.x.data[ problem.8.category.data == "Hexagon" ]

hexagon.y.vector <-
    problem.8.y.data[ problem.8.category.data == "Hexagon" ]
```

```
polygon(
    x = hexagon.x.vector,
    y = hexagon.y.vector,
    lty = "solid",
    lwd = 2,
    border = "cadetblue3"
)
```



Part (g): First rose curve

Now we'll draw the sixth figure of the mandala: an inscribed rose curve.

- First, copy your code from part (f).
- Select the x-values in `problem.8.x.data` that correspond to the elements in `problem.8.category.data` that are equal to "Rose Curve 1".
- Select the y-values in `problem.8.y.data` that correspond to the elements in `problem.8.category.data` that are equal to "Rose Curve 1".
- Then use the `polygon()` function to draw a connected sequence of line segments through the points in the filtered x- and y-values:
  - The line type should be "solid".
  - The line width should be 3.

- The line color should be “darkorchid4”.

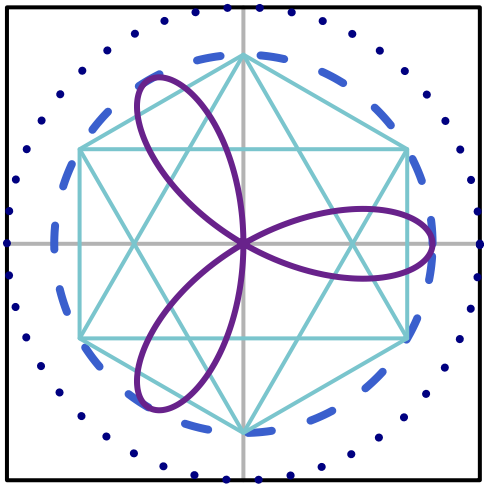
The polygon should not have a fill color, but instead should just have a black border. (The preceding sentence contains a subtle hint about the correct optional argument to use.) As a check, the light-grey lines that cross in the center of the picture should still be visible.

Solution

```
par( pty = "s" )

plot(
    x = NULL,
    xlim = c( -1, +1 ),
    ylim = c( -1, +1 ),
    xlab = "",
    ylab = "",
    axes = FALSE
)

segments(
    x0 = 0,
    y0 = -1,
    x1 = 0,
    y1 = +1,
    lty = "solid",
    lwd = 2,
    col = "gray70"
)



segments(
    x0 = -1,
    y0 = 0,
    x1 = +1,
    y1 = 0,
    lty = "solid",
    lwd = 2,
    col = "gray70"
)


polygon(
    x = c( 1, -1, -1, 1 ),
    y = c( 1, 1, -1, -1 ),
    lty = "solid",
    lwd = 2,
    border = "black"
)

outer.circle.x.vector <-
    problem.8.x.data[ problem.8.category.data == "Outer Circle" ]

outer.circle.y.vector <-
```

```
    problem.8.y.data[ problem.8.category.data == "Outer Circle" ]

lines(
    x = outer.circle.x.vector,
    y = outer.circle.y.vector,
    lty = "dotted",
    lwd = 4,
    col = "navy"
)


inner.circle.x.vector <-
    problem.8.x.data[ problem.8.category.data == "Inner Circle" ]

inner.circle.y.vector <-
    problem.8.y.data[ problem.8.category.data == "Inner Circle" ]

lines(
    x = inner.circle.x.vector,
    y = inner.circle.y.vector,
    lty = "dashed",
    lwd = 4,
    col = "royalblue3"
)


triangle.1.x.vector <-
    problem.8.x.data[ problem.8.category.data == "Triangle 1" ]

triangle.1.y.vector <-
    problem.8.y.data[ problem.8.category.data == "Triangle 1" ]

polygon(
    x = triangle.1.x.vector,
    y = triangle.1.y.vector,
    lty = "solid",
    lwd = 2,
    border = "cadetblue3"
)

triangle.2.x.vector <-
    problem.8.x.data[ problem.8.category.data == "Triangle 2" ]

triangle.2.y.vector <-
    problem.8.y.data[ problem.8.category.data == "Triangle 2" ]

polygon(
    x = triangle.2.x.vector,
    y = triangle.2.y.vector,
    lty = "solid",
    lwd = 2,
    border = "cadetblue3"
)
```

```
hexagon.x.vector <-
    problem.8.x.data[ problem.8.category.data == "Hexagon" ]

hexagon.y.vector <-
    problem.8.y.data[ problem.8.category.data == "Hexagon" ]

polygon(
    x = hexagon.x.vector,
    y = hexagon.y.vector,
    lty = "solid",
    lwd = 2,
    border = "cadetblue3"
)


rose.curve.1.x.vector <-
    problem.8.x.data[ problem.8.category.data == "Rose Curve 1" ]

rose.curve.1.y.vector <-
    problem.8.y.data[ problem.8.category.data == "Rose Curve 1" ]

polygon(
    x = rose.curve.1.x.vector,
    y = rose.curve.1.y.vector,
    lty = "solid",
    lwd = 3,
    border = "darkorchid4"
)
```



Part (h): Second rose curve

Now we'll draw the seventh figure of the mandala: another inscribed rose curve.

- First, copy your code from part (g).
- Select the x-values in `problem.8.x.data` that correspond to the elements in `problem.8.category.data` that are equal to "Rose Curve 2".
- Select the y-values in `problem.8.y.data` that correspond to the elements in `problem.8.category.data` that are equal to "Rose Curve 2".
- Then use the `polygon()` function to draw a connected sequence of line segments through the points in the filtered x- and y-values:
  - The line type should be "solid".
  - The line width should be 3.
  - The line color should be "darkorchid4".

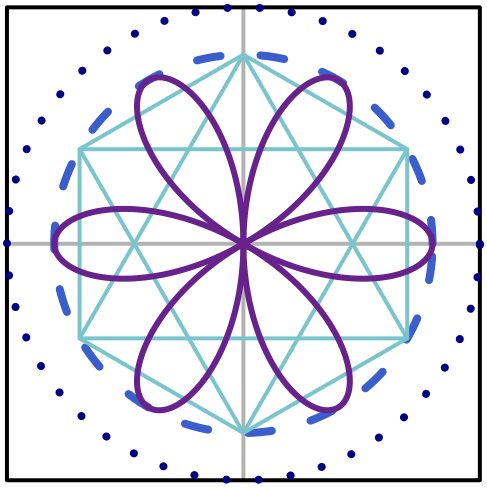
The polygon should not have a fill color, but instead should just have a black border. (The preceding sentence contains a subtle hint about the correct optional argument to use.) As a check, the light-grey lines that cross in the center of the picture should still be visible.

Solution

```
par( pty = "s" )

plot(
    x = NULL,
    xlim = c( -1, +1 ),
    ylim = c( -1, +1 ),
    xlab = "",
    ylab = "",
    axes = FALSE
)

segments(
    x0 = 0,
    y0 = -1,
    x1 = 0,
    y1 = +1,
    lty = "solid",
    lwd = 2,
    col = "gray70"
)



segments(
    x0 = -1,
    y0 = 0,
    x1 = +1,
    y1 = 0,
    lty = "solid",
    lwd = 2,
    col = "gray70"
)


polygon(
    x = c( 1, -1, -1, 1 ),
    y = c( 1, 1, -1, -1 ),
    lty = "solid",
    lwd = 2,
    border = "black"
)

outer.circle.x.vector <-
    problem.8.x.data[ problem.8.category.data == "Outer Circle" ]

outer.circle.y.vector <-
    problem.8.y.data[ problem.8.category.data == "Outer Circle" ]

lines(
    x = outer.circle.x.vector,
    y = outer.circle.y.vector,
    lty = "dotted",
    lwd = 4,
```

```
    col = "navy"
)


inner.circle.x.vector <-
    problem.8.x.data[ problem.8.category.data == "Inner Circle" ]

inner.circle.y.vector <-
    problem.8.y.data[ problem.8.category.data == "Inner Circle" ]

lines(
    x = inner.circle.x.vector,
    y = inner.circle.y.vector,
    lty = "dashed",
    lwd = 4,
    col = "royalblue3"
)


triangle.1.x.vector <-
    problem.8.x.data[ problem.8.category.data == "Triangle 1" ]

triangle.1.y.vector <-
    problem.8.y.data[ problem.8.category.data == "Triangle 1" ]

polygon(
    x = triangle.1.x.vector,
    y = triangle.1.y.vector,
    lty = "solid",
    lwd = 2,
    border = "cadetblue3"
)

triangle.2.x.vector <-
    problem.8.x.data[ problem.8.category.data == "Triangle 2" ]

triangle.2.y.vector <-
    problem.8.y.data[ problem.8.category.data == "Triangle 2" ]

polygon(
    x = triangle.2.x.vector,
    y = triangle.2.y.vector,
    lty = "solid",
    lwd = 2,
    border = "cadetblue3"
)


hexagon.x.vector <-
    problem.8.x.data[ problem.8.category.data == "Hexagon" ]

hexagon.y.vector <-
    problem.8.y.data[ problem.8.category.data == "Hexagon" ]
```

```
polygon(
    x = hexagon.x.vector,
    y = hexagon.y.vector,
    lty = "solid",
    lwd = 2,
    border = "cadetblue3"
)


rose.curve.1.x.vector <-
    problem.8.x.data[ problem.8.category.data == "Rose Curve 1" ]

rose.curve.1.y.vector <-
    problem.8.y.data[ problem.8.category.data == "Rose Curve 1" ]

polygon(
    x = rose.curve.1.x.vector,
    y = rose.curve.1.y.vector,
    lty = "solid",
    lwd = 3,
    border = "darkorchid4"
)


rose.curve.2.x.vector <-
    problem.8.x.data[ problem.8.category.data == "Rose Curve 2" ]

rose.curve.2.y.vector <-
    problem.8.y.data[ problem.8.category.data == "Rose Curve 2" ]

polygon(
    x = rose.curve.2.x.vector,
    y = rose.curve.2.y.vector,
    lty = "solid",
    lwd = 3,
    border = "darkorchid4"
)
```



Part (i): Emeralds

Now we'll draw the eighth figure of the mandala: precious emeralds!

- First, copy your code from part (h).
- Select the x-values in `problem.8.x.data` that correspond to the elements in `problem.8.category.data` that are equal to "Emeralds".
- Select the y-values in `problem.8.y.data` that correspond to the elements in `problem.8.category.data` that are equal to "Emeralds".
- Then use the `points()` function to draw points with coordinates in the filtered x- and y-values:
  - The point shape should be 22.
  - The border color of the point should be "black".
  - The width of the border should be 2.
  - The fill color should be "aquamarine2".

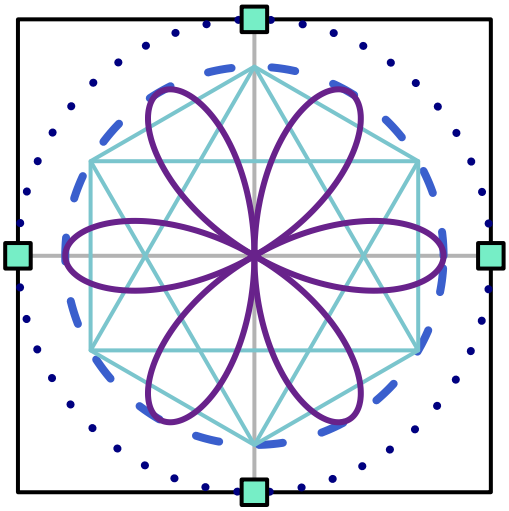
Solution

```
par( pty = "s" )

plot(
```

```
    x = NULL,
    xlim = c( -1, +1 ),
    ylim = c( -1, +1 ),
    xlab = "",
    ylab = "",
    axes = FALSE
)

segments(
    x0 = 0,
    y0 = -1,
    x1 = 0,
    y1 = +1,
    lty = "solid",
    lwd = 2,
    col = "gray70"
)



segments(
    x0 = -1,
    y0 = 0,
    x1 = +1,
    y1 = 0,
    lty = "solid",
    lwd = 2,
    col = "gray70"
)


polygon(
    x = c( 1, -1, -1, 1 ),
    y = c( 1, 1, -1, -1 ),
    lty = "solid",
    lwd = 2,
    border = "black"
)

outer.circle.x.vector <-
    problem.8.x.data[ problem.8.category.data == "Outer Circle" ]

outer.circle.y.vector <-
    problem.8.y.data[ problem.8.category.data == "Outer Circle" ]

lines(
    x = outer.circle.x.vector,
    y = outer.circle.y.vector,
    lty = "dotted",
    lwd = 4,
    col = "navy"
)
```

```
inner.circle.x.vector <-
    problem.8.x.data[ problem.8.category.data == "Inner Circle" ]

inner.circle.y.vector <-
    problem.8.y.data[ problem.8.category.data == "Inner Circle" ]

lines(
    x = inner.circle.x.vector,
    y = inner.circle.y.vector,
    lty = "dashed",
    lwd = 4,
    col = "royalblue3"
)


triangle.1.x.vector <-
    problem.8.x.data[ problem.8.category.data == "Triangle 1" ]

triangle.1.y.vector <-
    problem.8.y.data[ problem.8.category.data == "Triangle 1" ]

polygon(
    x = triangle.1.x.vector,
    y = triangle.1.y.vector,
    lty = "solid",
    lwd = 2,
    border = "cadetblue3"
)

triangle.2.x.vector <-
    problem.8.x.data[ problem.8.category.data == "Triangle 2" ]

triangle.2.y.vector <-
    problem.8.y.data[ problem.8.category.data == "Triangle 2" ]

polygon(
    x = triangle.2.x.vector,
    y = triangle.2.y.vector,
    lty = "solid",
    lwd = 2,
    border = "cadetblue3"
)


hexagon.x.vector <-
    problem.8.x.data[ problem.8.category.data == "Hexagon" ]

hexagon.y.vector <-
    problem.8.y.data[ problem.8.category.data == "Hexagon" ]

polygon(
    x = hexagon.x.vector,
    y = hexagon.y.vector,
```

```
    lty = "solid",
    lwd = 2,
    border = "cadetblue3"
)


rose.curve.1.x.vector <-
    problem.8.x.data[ problem.8.category.data == "Rose Curve 1" ]

rose.curve.1.y.vector <-
    problem.8.y.data[ problem.8.category.data == "Rose Curve 1" ]

polygon(
    x = rose.curve.1.x.vector,
    y = rose.curve.1.y.vector,
    lty = "solid",
    lwd = 3,
    border = "darkorchid4"
)


rose.curve.2.x.vector <-
    problem.8.x.data[ problem.8.category.data == "Rose Curve 2" ]

rose.curve.2.y.vector <-
    problem.8.y.data[ problem.8.category.data == "Rose Curve 2" ]

polygon(
    x = rose.curve.2.x.vector,
    y = rose.curve.2.y.vector,
    lty = "solid",
    lwd = 3,
    border = "darkorchid4"
)


emeralds.2.x.vector <-
    problem.8.x.data[ problem.8.category.data == "Emeralds" ]

emeralds.2.y.vector <-
    problem.8.y.data[ problem.8.category.data == "Emeralds" ]

points(
    x = emeralds.2.x.vector,
    y = emeralds.2.y.vector,
    pch = 22,
    lwd = 2,
    cex = 2,
    col = "black",
    bg = "aquamarine2"
)
```



Part (j): Sapphires, rubies, and a diamond

Now we'll draw the final figures of the mandala: sapphires, rubies, and even a diamond!

- First, copy your code from part (j).

Now we'll repeat the process that we used for the "Emeralds" category, using this table:

Category	Point shape	Point size	Fill Color
"Sapphires"	24	2	"royalblue1"
"Rubies"	25	2	"firebrick1"
"Diamond"	23	4	"cornsilk"

- Select the x-values in `problem.8.x.data` that correspond to the elements in `problem.8.category.data` that are equal to "Emeralds".
- Select the y-values in `problem.8.y.data` that correspond to the elements in `problem.8.category.data` that are equal to "Emeralds".
- Then use the `points()` function to draw points with coordinates in the filtered x- and y-values:
  - The point shape should be 22.
  - The border color of the point should be "black".
  - The width of the border should be 2.

– The fill color should be “aquamarine2”.

Solution

```
par( pty = "s" )

plot(
    x = NULL,
    xlim = c( -1, +1 ),
    ylim = c( -1, +1 ),
    xlab = "",
    ylab = "",
    axes = FALSE
)

segments(
    x0 = 0,
    y0 = -1,
    x1 = 0,
    y1 = +1,
    lty = "solid",
    lwd = 2,
    col = "gray70"
)



segments(
    x0 = -1,
    y0 = 0,
    x1 = +1,
    y1 = 0,
    lty = "solid",
    lwd = 2,
    col = "gray70"
)


polygon(
    x = c( 1, -1, -1, 1 ),
    y = c( 1, 1, -1, -1 ),
    lty = "solid",
    lwd = 2,
    border = "black"
)

outer.circle.x.vector <-
    problem.8.x.data[ problem.8.category.data == "Outer Circle" ]

outer.circle.y.vector <-
    problem.8.y.data[ problem.8.category.data == "Outer Circle" ]

lines(
    x = outer.circle.x.vector,
```

```
    y = outer.circle.y.vector,
    lty = "dotted",
    lwd = 4,
    col = "navy"
)


inner.circle.x.vector <-
    problem.8.x.data[ problem.8.category.data == "Inner Circle" ]

inner.circle.y.vector <-
    problem.8.y.data[ problem.8.category.data == "Inner Circle" ]

lines(
    x = inner.circle.x.vector,
    y = inner.circle.y.vector,
    lty = "dashed",
    lwd = 4,
    col = "royalblue3"
)


triangle.1.x.vector <-
    problem.8.x.data[ problem.8.category.data == "Triangle 1" ]

triangle.1.y.vector <-
    problem.8.y.data[ problem.8.category.data == "Triangle 1" ]

polygon(
    x = triangle.1.x.vector,
    y = triangle.1.y.vector,
    lty = "solid",
    lwd = 2,
    border = "cadetblue3"
)

triangle.2.x.vector <-
    problem.8.x.data[ problem.8.category.data == "Triangle 2" ]

triangle.2.y.vector <-
    problem.8.y.data[ problem.8.category.data == "Triangle 2" ]

polygon(
    x = triangle.2.x.vector,
    y = triangle.2.y.vector,
    lty = "solid",
    lwd = 2,
    border = "cadetblue3"
)


hexagon.x.vector <-
    problem.8.x.data[ problem.8.category.data == "Hexagon" ]
```

```
hexagon.y.vector <-
    problem.8.y.data[ problem.8.category.data == "Hexagon" ]

polygon(
    x = hexagon.x.vector,
    y = hexagon.y.vector,
    lty = "solid",
    lwd = 2,
    border = "cadetblue3"
)


rose.curve.1.x.vector <-
    problem.8.x.data[ problem.8.category.data == "Rose Curve 1" ]

rose.curve.1.y.vector <-
    problem.8.y.data[ problem.8.category.data == "Rose Curve 1" ]

polygon(
    x = rose.curve.1.x.vector,
    y = rose.curve.1.y.vector,
    lty = "solid",
    lwd = 3,
    border = "darkorchid4"
)


rose.curve.2.x.vector <-
    problem.8.x.data[ problem.8.category.data == "Rose Curve 2" ]

rose.curve.2.y.vector <-
    problem.8.y.data[ problem.8.category.data == "Rose Curve 2" ]

polygon(
    x = rose.curve.2.x.vector,
    y = rose.curve.2.y.vector,
    lty = "solid",
    lwd = 3,
    border = "darkorchid4"
)


emeralds.2.x.vector <-
    problem.8.x.data[ problem.8.category.data == "Emeralds" ]

emeralds.2.y.vector <-
    problem.8.y.data[ problem.8.category.data == "Emeralds" ]

points(
    x = emeralds.2.x.vector,
    y = emeralds.2.y.vector,
    pch = 22,
    lwd = 2,
```

```
    cex = 2,
    col = "black",
    bg = "aquamarine2"
)

sapphires.2.x.vector <-
    problem.8.x.data[ problem.8.category.data == "Sapphires" ]

sapphires.2.y.vector <-
    problem.8.y.data[ problem.8.category.data == "Sapphires" ]

points(
    x = sapphires.2.x.vector,
    y = sapphires.2.y.vector,
    pch = 24,
    lwd = 2,
    cex = 2,
    col = "black",
    bg = "royalblue1"
)

rubies.2.x.vector <-
    problem.8.x.data[ problem.8.category.data == "Rubies" ]

rubies.2.y.vector <-
    problem.8.y.data[ problem.8.category.data == "Rubies" ]

points(
    x = rubies.2.x.vector,
    y = rubies.2.y.vector,
    pch = 21,
    lwd = 2,
    cex = 1.5,
    col = "black",
    bg = "firebrick1"
)

diamond.2.x.vector <-
    problem.8.x.data[ problem.8.category.data == "Diamond" ]

diamond.2.y.vector <-
    problem.8.y.data[ problem.8.category.data == "Diamond" ]

points(
    x = diamond.2.x.vector,
    y = diamond.2.y.vector,
    pch = 23,
    lwd = 2,
    cex = 4,
    col = "black",
    bg = "cornsilk1"
)
```