Problem Set 7 Solutions

CSCI 5a: Programming in R

Let's clear the global computing environment:

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

```
load( "Problem Set 7 R Objects.Rdata" )
```

Let's review the objects that we just loaded in:

```
ls()
```

```
## [1] "problem.3.number.of.shares.sold.data"
## [2] "problem.3.price.per.share.data"
## [3] "problem.4.data"
## [4] "problem.5.cereal.brand.data"
## [5] "problem.6.number.of.widgets.sold.data"
## [6] "problem.6.region.data"
```

Problem 1: Baseball Functions

Of course I'm going to ask you this.

Part (a): Batting average

Write a function that takes two arguments, the number of at-bats and the number of hits, and returns a numeric value representing the player's batting average.

Then use this function to calculate Ted Williams' batting average. Report your result using a `cat()` statement, displaying this value using the standard baseball convention i.e. no leading 0 and with three digits to the right of the decimal place. (See Exercise 1.1 in Module 1 for how to do this.)

Solution

```
batting.average <-
    function( at.bats, hits ) {
        return(
            hits / at.bats
        )
    }
```

From Lecture 2, Module 3, Exercise 2.1, we know Ted Williams' baseball statistics:

Statistics	Value
Plate appearances	9,792
At-bats	7,706
Hits	2,654
Doubles	525
Triples	71
Home Runs	521
Bases on balls	2,021
Hit by a pitch	39
Sacrifice flies	20

```
ted.williams.batting.average <-
    batting.average(
        at.bats = 7706,
        hits = 2654
    )

cat(
    "Ted Williams batting average:",
    substr(
        formatC(
            ted.williams.batting.average,
            format = "f",
            digits = 3
        ),
        start = 2,
        stop = 5
    )
)
```

```
## Ted Williams batting average: .344
```

Part (b): On-Base percentage

Write a function that takes five input arguments:

- The number of at-bats
- The number of hits
- The number of bases on balls
- The number of times hit by a pitch
- The number of sacrifice flies

The function then returns a numeric value for the on-base percentage (using the simplified definition for our course).

Then use this function to calculate Willie Mays' on-base percentage. Report your result using a `cat()` statement, displaying this value using the standard baseball convention.

Solution

```
on.base.percentage <-
    function(
        at.bats,
        hits,
        bases.on.balls,
        hit.by.pitch,
        sacrifice.flies
    ) {
        return(
            (hits + bases.on.balls + hit.by.pitch) /
                (at.bats + bases.on.balls + hit.by.pitch + sacrifice.flies)
        )
    }
```

From Problem Set 3, Problem 1, we have:

Statistics	Value
Plate appearances	12,497
At-bats	10,881
Hits	3,283
Doubles	523
Triples	140
Home Runs	660
Bases on balls	1,464
Hit by a pitch	44
Sacrifice flies	91

Then Willie Mays' on-base percentage is:

```
willie.mays.on.base.percentage <-
    on.base.percentage(
        at.bats = 10881,
        hits = 3283,
        bases.on.balls = 1464,
        hit.by.pitch = 44,
        sacrifice.flies = 91
    )
```

```
cat(
    "Willie Mays' on-base percentage:",
    substr(
        formatC(
            willie.mays.on.base.percentage,
            format = "f",
            digits = 3
        ),
        start = 2,
        stop = 5
    )
)
```

```
## Willie Mays' on-base percentage: .384
```

Part (c): Slugging percentage

Write a function that takes five arguments:

- The number of at-bats
- The number of hits
- The number of doubles
- The number of triples
- The number of home runs

The function then returns a numeric value for the slugging percentage.

Then use this function to calculate Babe Ruth's slugging percentage. Report your result using a `cat()` statement, displaying this value using the standard baseball convention.

Solution

From Lecture 2, Module 3, Section 5, we have Babe Ruth's baseball statistics:

Statistics	Value
Plate appearances	10,626
At-bats	8,399
Hits	2,873
Doubles	506
Triples	136
Home runs	714
Bases on balls	2,062
Hit by a pitch	43
Sacrifice flies	0

```
slugging.percentage <-
    function(
        at.bats,
        hits,
        doubles,
        triples,
        home.runs
    ) {
        singles <-
            hits -
            (doubles + triples + home.runs)

        total.bases <-
            (1 * singles) +
            (2 * doubles) +
            (3 * triples) +
            (4 * home.runs)

        return(
            total.bases / at.bats
        )
    }
```

We know from Lecture 3, Module 3, that Babe Ruth had 8,399 at-bats, 2,873 hits, 506 doubles, 136 triples, and 714 home runs.

```
babe.ruth.slugging.percentage <-
    slugging.percentage(
        at.bats = 8399,
        hits = 2873,
        doubles = 506,
        triples = 136,
        home.runs = 714
    )
```

```
cat(
    "Babe Ruth's slugging percentage:",
    substr(
        formatC(
            babe.ruth.slugging.percentage,
            format = "f",
            digits = 3
        ),
        start = 2,
        stop = 5
    )
)
```

```
## Babe Ruth's slugging percentage: .690
```

End of problem 1

Problem 2: Baseball Reporter

Now we can use the functions that you wrote in Problem 1 to help build a baseball reporter.

Part (a): Reporter function

Write a reporter function that takes 9 input arguments:

- The player's name
- The number of at-bats
- The number of hits
- The number of doubles
- The number of triples
- The number of home runs
- The number of bases on balls
- The number of times hit by a pitch
- The number of sacrifice flies

The function then prints out a report on the player:

- The report first lists the player's name.
- The report then displays a tabulation of the player's basic statistics:
  - The number of at-bats
  - The number of outs
  - The number of bases on balls
  - The number of hits
  - The number of singles
  - The number of doubles
  - The number of triples
  - The number of home runs
- Then the report lists the three standard batting performance statistics:
  - Batting average
  - On-base percentage
  - Slugging percentage

All of these perfomance statistics should be reported with a `cat()` statement using the standard baseball convention.

There's nothing to report here, but write your code clearly so the TAs can understand what you're doing.

Solution

```
baseball.reporter <-
    function(
        player.name,
        at.bats,
        hits,
        doubles,
        triples,
        home.runs,
        bases.on.balls,
        hit.by.pitch,
        sacrifice.flies
    ) {
        outs <-
            at.bats - hits

        singles <-
            hits - (doubles + triples + home.runs)

        player.batting.average <-
            batting.average(
                at.bats = at.bats,
                hits = hits
            )

        player.on.base.percentage <-
            on.base.percentage(
                at.bats = at.bats,
                hits = hits,
                bases.on.balls = bases.on.balls,
                hit.by.pitch = hit.by.pitch,
                sacrifice.flies = sacrifice.flies
            )

        player.slugging.percentage <-
            slugging.percentage(
                at.bats = at.bats,
                hits = hits,
                doubles = doubles,
                triples = triples,
                home.runs = home.runs
            )

        cat(
            "Player name:",
            player.name,
            "\n\n"
        )

        cat(
            "\tBasic statistics:\n"
        )

        cat(
```

```
        "\t\tAt-bats:",
        at.bats,
        "\n"
    )

    cat(
        "\t\tOuts:",
        outs,
        "\n"
    )

    cat(
        "\t\tBases on balls:",
        bases.on.balls,
        "\n"
    )

    cat(
        "\t\tHits:",
        hits,
        "\n"
    )

    cat(
        "\t\tSingles:",
        singles,
        "\n"
    )

    cat(
        "\t\tDoubles:",
        doubles,
        "\n"
    )

    cat(
        "\t\tTriples:",
        triples,
        "\n"
    )

    cat(
        "\t\tHome runs:",
        home.runs,
        "\n\n"
    )

    cat(
        "\tBatting average:",
        substr(
            formatC(
                player.batting.average,
                format = "f",
```

```
                digits = 3
            ),
        start = 2,
        stop = 5
        ),
        "\n"
    )

    cat(
        "\tOn-base percentage:",
        substr(
            formatC(
                player.on.base.percentage,
                format = "f",
                digits = 3
            ),
        start = 2,
        stop = 5
        ),
        "\n"
    )

    cat(
        "\tSlugging percentage:",
        substr(
            formatC(
                player.slugging.percentage,
                format = "f",
                digits = 3
            ),
        start = 2,
        stop = 5
        ),
        "\n"
    )
}
```

Part (b): Hank Aaron

Use the function you constructed in part (a) to generate a batting performance report for Hank Aaron.

Statistics	Value
Plate appearances	13,941
At-bats	12,364
Hits	3,771
Doubles	624
Triples	98
Home Runs	755
Bases on balls	1,402
Hit by a pitch	32
Sacrifice flies	121

Solution

```
baseball.reporter(
    "Hank Aaron",
    at.bats = 12364,
    hits = 3771,
    doubles = 624,
    triples = 98,
    home.runs = 755,
    bases.on.balls = 1402,
    hit.by.pitch = 32,
    sacrifice.flies = 121
)
```

```
## Player name: Hank Aaron
##
##  Basic statistics:
##      At-bats: 12364
##      Outs: 8593
##      Bases on balls: 1402
##      Hits: 3771
##      Singles: 2294
##      Doubles: 624
##      Triples: 98
##      Home runs: 755
##
##  Batting average: .305
##  On-base percentage: .374
##  Slugging percentage: .555
```

End of problem 2

Problem 3: VWAP

Part (a): Function definition

Suppose we have data on a number of stock sales for a particular company, and for each sale we know the price per share and the number of shares sold.

Write a function that takes two input arguments:

- The first argument is a numeric vector consisting of the price per share for each transaction.
- The second argument is a numeric vector consisting of the number of shares sold for each transaction.

The function then returns a numeric value for the VWAP.

There's nothing to report here, but write your code clearly so the TAs can understand what you're doing.

Solution

```
vwap <-
  function(
    price.per.share.vector,
    number.of.shares.sold.vector
  ) {

    total.sales.amount <-
      sum( price.per.share.vector * number.of.shares.sold.vector )

    total.number.of.shares.sold <-
      sum( number.of.shares.sold.vector )

    vwap <-
      total.sales.amount / total.number.of.shares.sold

    return( vwap )

}
```

Part (b): Calculation

The vector `problem.3.price.per.share.data` contain the price per share for 185 stock transactions. The vector `problem.3.number.of.shares.sold.data` contains data on the number of shares sold in each of the corresponding transactions.

Use the function that you defined in part (a) to calculate the VWAP for this stock. Report your result using a `cat()` statement, displaying this value with 2 decimal places.

Solution

```
problem.3.vwap <-
    vwap(
        price.per.share.vector =
          problem.3.price.per.share.data,
        number.of.shares.sold.vector =
          problem.3.number.of.shares.sold.data
```

```
    )

cat(
    "VWAP:",
    formatC(
        problem.3.vwap,
        format = "f",
        digits = 2
    )
)
```

```
## VWAP: 72.64
```

End of problem 3

Problem 4: Three Views

In this problem, we'll examine the same set of data using three different visualization methods.

For each graph, produce a finished version with all the features that we've studied e.g. titles, colors, point shapes, jitter, etc.

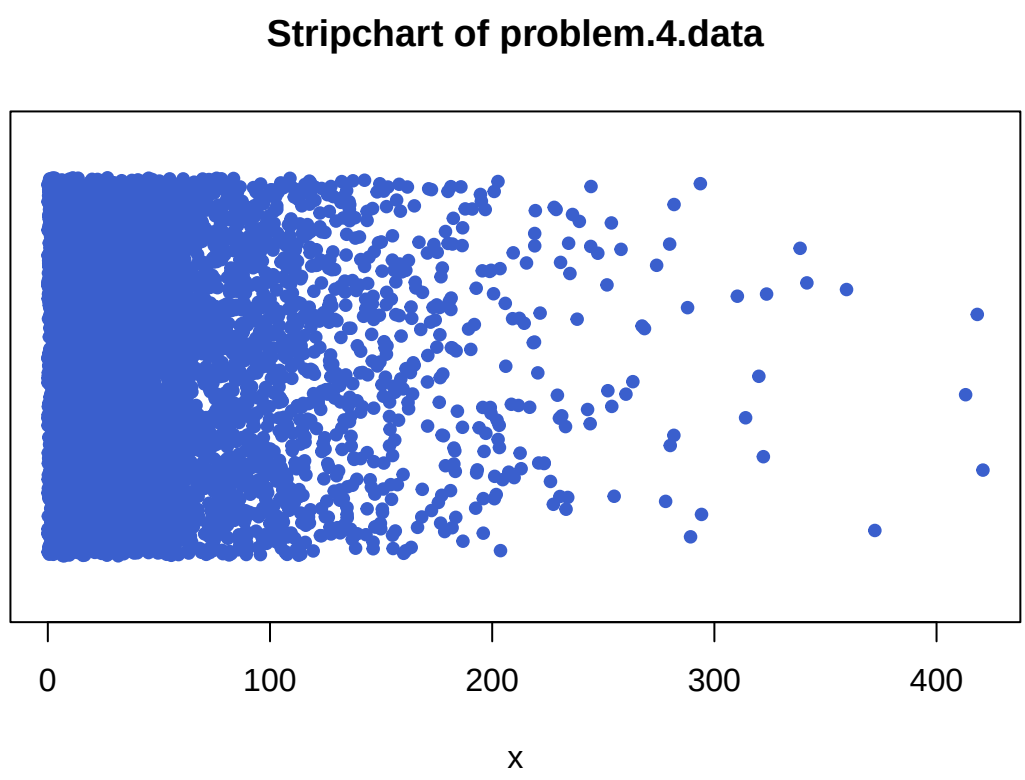
When you finish the third graph, you should go back and compare all three graphs. Which kinds of information are displayed by each chart? Is there one display that you think is particularly useful? Which is your personal favorite? You don't have to write out answer, but I think you'll get a lot more out of this problem if you engage with it and try to draw some conclusions for yourself.

Part (a): Stripchart

Construct a stripchart for the values in `problem.4.data`.

Solution

```
stripchart(
    problem.4.data,
    ylim = c(0, 2),
    main = "Stripchart of problem.4.data",
    xlab = "x",
    method = "jitter",
    jitter = 0.8,
    pch = 19,
    cex = 0.8,
    col = "royalblue3"
)
```

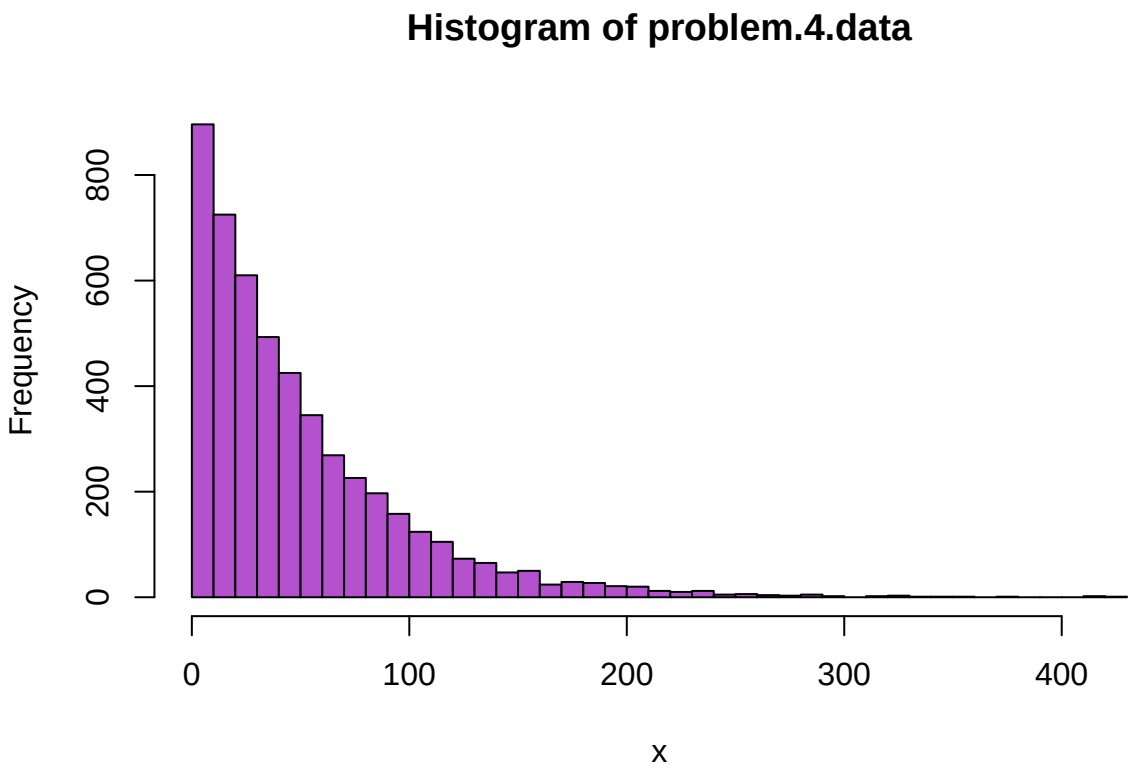


Part (b): Histogram

Construct a histogram for the data in `problem.4.data`.

Solution

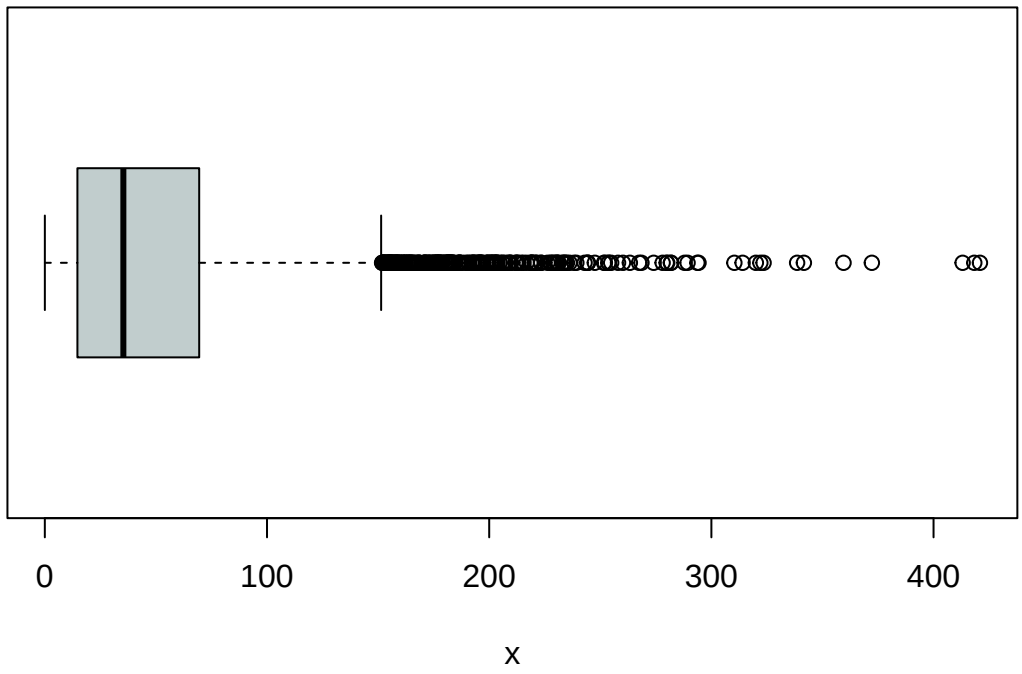
```
hist(
    x = problem.4.data,
    main = "Histogram of problem.4.data",
    xlab = "x",
    breaks = 50,
    col = "mediumorchid3"
)
```



Part (c): Boxplot

Construct a boxplot for the values in `problem.4.data`.

```
boxplot(
    x = problem.4.data,
    main = "Boxplot of Problem 4 Data",
    xlab = "x",
    horizontal = TRUE,
    col = "azure3"
)
```


Boxplot of Problem 4 Data
0
100
200
300
400
x

End of Problem 4

Problem 5: Removing Internal Codes, Part 1

The character string vector `problem.5.cereal.brand.data` contains data on each box of cereal that is sold.

Each item in this vector consists of an internal sales code and an identifier for the cereal brand.

The internal sales code has a fixed length, and the identifier for the cereal brand is always 3 letters.

For this problem, you should construct a frequency count table of the cereal brands and then display this in a barplot with the levels organized in decreasing order.

You're on your own for this one! You'll have to figure out how to do this, and there are alternative approaches. Just make sure that you show us the frequency count table and the barplot with the levels organized in decreasing frequency, and make them look nice so the TAs can grade them.

Solution

Let's start by taking a quick look at the data:

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

```
## [1] "338865-KYM" "411866-KYM" "590882-SBZ" "531017-SBZ" "393852-SBZ"
## [6] "476339-KYM" "171010-SBZ" "581362-SBZ"
```

By inspection, the internal code is 7 characters long, followed by the three-character cereal brand abbreviation.

Let's strip out this internal code:

```
stripped.cereal.brand.vector <-
  substr(
    x = problem.5.cereal.brand.data,
    start = 8,
    stop = 10
  )

head( stripped.cereal.brand.vector, n = 8 )
```

```
## [1] "KYM" "KYM" "SBZ" "SBZ" "SBZ" "KYM" "SBZ" "SBZ"
```

So now we've removed the internal codes.

Let's take a look at the values in `stripped.cereal.brand.vector`:

```
unique( stripped.cereal.brand.vector )
```

```
##  [1] "KYM" "SBZ" "Kym" "HKT" "Sbz" "sbz" "kym" "hKt" "hkt" "sbZ" "hkT"
```

```
toupper( unique( stripped.cereal.brand.vector ) )
```

```
##  [1] "KYM" "SBZ" "KYM" "HKT" "SBZ" "SBZ" "KYM" "HKT" "HKT" "SBZ" "HKT"
```

We'll have to clean this up.

Let's use a lookup vector:

```
cereal.brand.lookup.vector <-
  c(
    "SBZ" = "SBZ",
    "HKT" = "HKT",
    "hkt" = "HKT",
    "KYM" = "KYM",
    "hKt" = "HKT",
    "Sbz" = "SBZ",
    "kym" = "KYM",
    "Kym" = "KYM",
    "sbz" = "SBZ",
    "sbZ" = "SBZ",
    "hkT" = "HKT"
  )
```

Now we can group the levels:

```
cleaned.cereal.brand.vector <-
  cereal.brand.lookup.vector[
    stripped.cereal.brand.vector
  ]

head( cleaned.cereal.brand.vector )
```

```
##   KYM   KYM   SBZ   SBZ   SBZ   KYM
## "KYM" "KYM" "SBZ" "SBZ" "SBZ" "KYM"
```

Let's check the levels:

```
unique( cleaned.cereal.brand.vector )
```

```
## [1] "KYM" "SBZ" "HKT"
```

Now let's make a frequency count table of this data:

```
cereal.brand.frequency.count.table <-
  table( cleaned.cereal.brand.vector )

cereal.brand.frequency.count.table
```

```
## cleaned.cereal.brand.vector
## HKT KYM SBZ
## 185 249 487
```

Notice that the levels are not in decreasing order, so we'll have to sort the table:

```
sorted.cereal.brand.frequency.count.table <-
  sort(
    cereal.brand.frequency.count.table, decreasing = TRUE
  )

sorted.cereal.brand.frequency.count.table
```

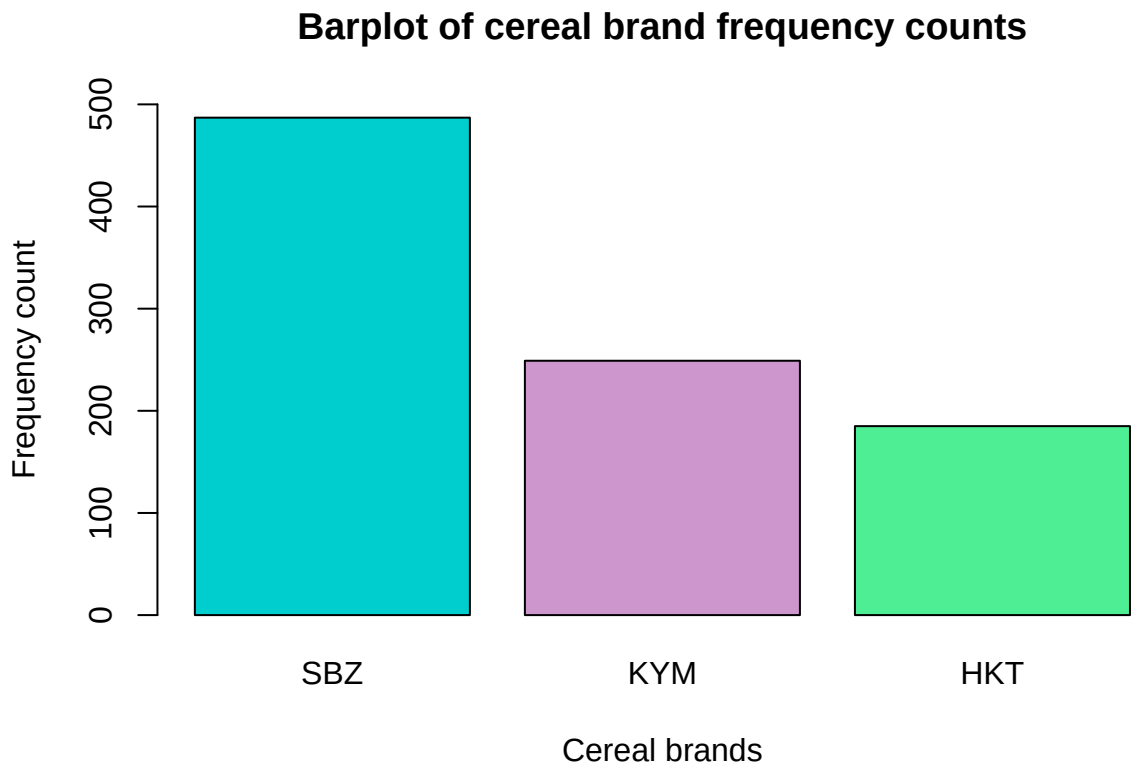
```
## cleaned.cereal.brand.vector
## SBZ KYM HKT
## 487 249 185
```

Now let's display this in a barplot:

```
sorted.cereal.brand.frequency.count.table
```

```
## cleaned.cereal.brand.vector
## SBZ KYM HKT
## 487 249 185
```

```
barplot(
  height = sorted.cereal.brand.frequency.count.table,
  ylim = c(0, 500),
  main = "Barplot of cereal brand frequency counts",
  xlab = "Cereal brands",
  ylab = "Frequency count",
  col = c( "cyan3", "plum3", "seagreen2" )
)
```



End of Problem 5

Problem 6: Removing Internal Codes, Part 2

WiDgT has data on all the transactions for their retail widget sales across four regions: North, East, South, and West.

The character string vector `problem.6.region.data` contains the region for each transaction, but preceded by an internal code of fixed length.

The vector `problem.6.number.of.widgets.sold.data` contains the number of widgets sold in the transaction.

In this problem, you must construct a table of relative proportions for the total number of widgets sold across the four regions, formatting all values with 2 decimal places.

Thus, if there were 1,000 total widgets sold across all 4 regions, and 400 were sold in the North region, then in the table of relative proportions you should display the value 0.40 for North.

In the table, the four regions should be labelled "North", "East", "South", and "West", and should be presented in that order.

Part (a): Stripping out the internal codes

Create a new vector consisting of the values of `problem.6.region.data` with the internal codes stripped out.

The problem statement only specified that the internal codes all have the same fixed length, but did not specify this length, so you'll have to do some exploration to determine how long it is.

Then create a factor from this new vector, with the correct region labels and order, and save this factor in a variable.

Finally, directly display a frequency count table of the regions so the TAs can check your work.

Solution

First, let's take a look at the data:

```
head( problem.6.region.data )
```

```
## [1] "664319320-north" "947356409-south" "949486926-north" "718161525-south"
## [5] "631157340-south" "505423506-north"
```

So it seems that the internal code is 9 digits followed by a hyphen.

Let's strip out these internal codes:

```
stripped.region.vector <-
  substr(
    x = problem.6.region.data,
    start = 11,
    stop = nchar( problem.6.region.data )
  )

head( stripped.region.vector, n = 10 )
```

```
##  [1] "north" "south" "north" "south" "south" "north" "north" "west"  "north"
## [10] "north"
```

Actually, we don't need to be so precise about getting the end right.

All we need are the first few letters to determine the region, and then we can fix the labels when we make the factor.

So we could just do this:

```
stripped.region.vector <-
  substr(
    x = problem.6.region.data,
    start = 11,
    stop = 13
  )

head( x = stripped.region.vector, n = 8 )
```

```
## [1] "nor" "sou" "nor" "sou" "sou" "nor" "nor" "wes"
```

Let's check the levels:

```
unique( stripped.region.vector )
```

```
## [1] "nor" "sou" "wes" "eas"
```

So we don't have any bad values that we need to clean.

Now we can construct the factor with the proper label names and ordering:

```
region.factor <-
  factor(
    x = stripped.region.vector,
    levels = c( "nor", "eas", "sou", "wes" ),
    labels = c( "North", "East", "South", "West" )
  )

table( region.factor )
```

```
## region.factor
## North  East South  West
##   387   156   202    84
```

Part (b): North region total sales

Now let's review how to calculate the total number of widgets sold in a region.

As an example, we'll focus on the North region.

- First, construct a logical indexing vector where an element is `TRUE` if the corresponding element of the stripped region data from Part (a) represents the North region.
- Use this logical indexing vector to select the elements of `problem.8.number.of.widgets.sold.data` that were sold in the North district.

- Add up the values of the North widget sales data to obtain the total sales amount, and store this in a variable.

Report your final result using a `cat()` statement, formatting the value with 0 decimal places.

Solution

First, let's create the logical indexing vector:

```
north.region.logical.indexing.vector <-
  region.factor == "North"
```

Now let's use this logical indexing vector to select the elements of `problem.6.number.of.widgets.sold.data` that were sold in the North region:

```
north.number.of.widgets.sold.vector <-
  problem.6.number.of.widgets.sold.data[
    north.region.logical.indexing.vector
  ]
```

We calculate the total number of widgets sold by adding up the values in this filtered vector:

```
north.total.number.of.widgets.sold <-
  sum( north.number.of.widgets.sold.vector )
```

In fact, we could have done this:

```
north.total.number.of.widgets.sold <-
  sum(
    problem.6.number.of.widgets.sold.data[
      region.factor == "North"
    ]
  )
```

Now we can report this:

```
cat(
  "North region total number of widgets sold:",
  formatC(
    north.total.number.of.widgets.sold,
    format = "f",
    digits = 0
  )
)
```

```
## North region total number of widgets sold: 1303
```

Part (c): Total sales function

We could repeat the calculations we just did to obtain the total number of widgets sold for the East, South, and West regions.

That would involve a lot of redundant code.

Instead, let's bundle this computation into a function, which will make our code much cleaner.

Write a function that takes a region name as the input argument and returns the total widget sales for that region.

You can assume that the factor that you created in Part (a) and `problem.6.number.of.widgets.sold.data` are global variables.

There are many ways to do this, but here's one suggestion:

- First construct a logical indexing vector that is TRUE when the corresponding element of `problem.6.region.data` is equal to the input argument region name and FALSE otherwise, and save this in a variable.
- Use this logical indexing vector to select the elements of `problem.6.number.of.widgets.sold.data` that occurred in the specified region, and save this vector in a variable.
  - Add the elements of this filtered vector, and save this in a variable.
  - Return the value of this sum.

There are more compressed ways to do this, but it's fine (and good!) to break the function into a sequence of intermediate steps.

There's nothing to report here, but write your code clearly so the TAs can understand what you're doing.

Solution

```
total.regional.number.of.widgets.sold <-
  function( region ) {

    logical.indexing.vector <-
      region.factor == region

    region.widget.sales.vector <-
      problem.6.number.of.widgets.sold.data[
        logical.indexing.vector
      ]

    total.sales <-
      sum( region.widget.sales.vector )

    return( total.sales )
  }
```

Part (d): North total sales

Use the function you defined in Part (c) to calculate the total sales for the North region. Store your result in a variable and report it using a `cat()` statement, displaying the value with 2 decimal places.

Hint: You should get the same value as you did in Part (b).

Solution

```
north.total.number.of.widgets.sold <-
  total.regional.number.of.widgets.sold( "North" )

cat(
  "North total sales:",
  formatC(
    x = north.total.number.of.widgets.sold,
    format = "f",
    digits = 0
  )
)
```

```
## North total sales: 1303
```

Part (e): East total sales

Use the function you defined in Part (c) to calculate the total sales for the East region. Store your result in a variable and report it using a `cat()` statement, displaying the value with 2 decimal places.

Solution

```
east.total.number.of.widgets.sold <-
  total.regional.number.of.widgets.sold( "East" )

cat(
  "East total sales:",
  formatC(
    x = east.total.number.of.widgets.sold,
    format = "f",
    digits = 0
  )
)
```

```
## East total sales: 543
```

Part (f): South total sales

Use the function you defined in Part (c) to calculate the total sales for the South region. Store your result in a variable and report it using a `cat()` statement, displaying the value with 2 decimal places.

Solution

```
south.total.number.of.widgets.sold <-
  total.regional.number.of.widgets.sold( "South" )

cat(
  "South total sales:",
  formatC(
    x = south.total.number.of.widgets.sold,
    format = "f",
    digits = 0
  )
)
```

```
## South total sales: 729
```

Part (g): West total sales

Use the function you defined in Part (c) to calculate the total sales for the West region. Store your result in a variable and report it using a `cat()` statement, displaying the value with 2 decimal places.

Solution

```
west.total.number.of.widgets.sold <-
  total.regional.number.of.widgets.sold( "West" )

cat(
  "West total sales:",
  formatC(
    x = south.total.number.of.widgets.sold,
    format = "f",
    digits = 0
  )
)
```

```
## West total sales: 729
```

Part (h): Named vector

Construct a named vector where the names are the four regions in correct order, and the corresponding values of the named vector are the regional total number of widgets sold that we've calculated in parts (d) through (g).

Save the named vector in a variable, and display it directly for the TAs to check your work.

Solution

```
regional.total.number.of.widgets.sold.vector <-
  c(
    "North" = north.total.number.of.widgets.sold,
    "East" = east.total.number.of.widgets.sold,
    "South" = south.total.number.of.widgets.sold,
    "West" = west.total.number.of.widgets.sold
  )

regional.total.number.of.widgets.sold.vector
```

```
## North  East South  West
##  1303   543   729   272
```

Part (i): Relative proportions table

Finally, we can construct a table of the relative proportions of the total widget sales across the four regions.

Recall that the `proportions()` function takes one input argument, which must be a table.

In part (h), we constructed a named vector, not a table.

Use the `as.table()` function to convert the named vector in part (h) to a table, and then use this table to construct a table of the relative proportions of the total number of widgets sold across the four regions.

Directly display the relative proportions table for the TAs to grade.

Solution

Now we can convert this named vector to a table by using the `as.table()` function:

```
regional.total.number.of.widgets.sold.table <-
  as.table( regional.total.number.of.widgets.sold.vector )

regional.total.number.of.widgets.sold.table
```

```
## North  East South  West
##  1303   543   729   272
```

Finally, we can construct the table of proportions:

```
regional.total.number.of.widgets.sold.relative.proportions.table <-
  round(
    proportions( regional.total.number.of.widgets.sold.table ),
    digits = 2
  )

regional.total.number.of.widgets.sold.relative.proportions.table
```

```
## North  East South  West
##  0.46  0.19  0.26  0.10
```

```
input.vector <-
  c( "abcd", "pqr", "a4rgz", "wxyz" )

first.letter.vector <-
  substr( input.vector, start = 1, stop = 1)

first.letter.a.logical.vector <-
  first.letter.vector == "a"

input.vector[ first.letter.a.logical.vector ]
```

```
## [1] "abcd"  "a4rgz"
```

```
input.vector[ first.letter.vector == "a" ]
```

```
## [1] "abcd"  "a4rgz"
```

```
p.w.vector <- c("p", "w")

input.vector[
  substr( input.vector, start = 1, stop = 1) %in% p.w.vector
]
```

```
## [1] "pqr"  "wxyz"
```

```
7/6 * 4/3
```

```
## [1] 1.555556
```