Problem Set 8 Solutions

CSCI E-5a: Programming in R

Problem 1: Data Frame Reporter

Throughout this course, we've written what I call "reporter" functions, which take an R object and print out a range of information about its properties. Reporter functions are extremely useful when we first read in a data set, because they can quickly summarize a number of features of the data.

For this problem, you must construct a reporter function for a data frame. The reporter function takes one argument, a data frame, and then prints out a range of information:

- First, it reports the total number of rows and columns in the data frame.
- Then the reporter function prints out information for each column:
  - First, it prints out the name of the column.
  - Next, it prints out the class of the column, either `numeric`, `logical`, or `character`.
  - If the column is a numeric vector, then the reporter function prints out the sample mean, sample standard deviation, sample minimum, and sample maximum.
  - If the column is a logical vector, then the reporter function prints out the proportion of elements that are true.
  - If the column is a character string vector, then the reporter function prints out the number of unique values, as well as the first 3 unique values.
  - Finally, for each column, the reporter function prints out the number of missing items.

Part (a): Data frame reporter function

Write a data frame reporter function according the specifications above. Make the output nice by using newlines and tab characters to space out the display. There's nothing to report here, but write you code clearly so the TAs can understand what you're doing.

Solution

```
data.frame.reporter <- function( input.data.frame ) {

    number.of.columns <- ncol( input.data.frame )

    number.of.rows <- nrow( input.data.frame )

    column.names.vector <- names( input.data.frame )

    cat( "Number of columns:", number.of.columns, "\n" )

    cat( "Number of rows:", number.of.rows, "\n" )
```

```
for( column.name in column.names.vector ) {

    current.column <-
        input.data.frame[[ column.name ]]

    cat( "\nColumn name:", column.name, "\n" )

    column.class <-
        class( current.column )

    cat( "Column class:", column.class, "\n" )

    if( is.numeric( current.column ) ) {

        cat(
            "\tSample mean:",
            round(
                mean( current.column, na.rm = TRUE ),
                2
            ),
            "\n"
        )

        cat( "\tSample standard deviation:",
             round( sd( current.column, na.rm = TRUE ), 2 ), "\n" )

        cat( "\tSample minimum:",
             round( min( current.column, na.rm = TRUE ), 2 ), "\n" )

        cat( "\tSample maximum:",
             round( max( current.column, na.rm = TRUE ), 2 ), "\n" )

    } else if( is.logical( current.column ) ) {

            cat( "\tProportion true:",
                 mean( current.column, na.rm = TRUE ), "\n" )

    } else {

        unique.values <-
            unique( current.column )

        cat( "\tNumber of unique values:",
             length( unique.values ), "\n" )

        for( index in 1:3 ) {

            cat( "\t\t",
                 index,
                 ". ",
                 unique.values[ index ],
                 "\n",
                 sep = "" )
```

```
            }
        }

        cat( "\n\tNumber of missing items:",
             sum( is.na( current.column ) ),
             "\n" )
    }
}
```

Part (b): Testing the data frame reporter

Read in the data from the file "Problem 1 Data.csv" located in the "Problem Set 8 Data" folder and run your data frame reporter on it.

Solution

First, let's read in the data frame, and store it in a variable:

```
problem.1.data.frame <-
    read.csv(
        "../Problem Set 8 Data/Problem 1 Data.csv"
    )
```

Now we can run the data.frame.reporter() function:

```
data.frame.reporter( problem.1.data.frame )
```

```
## Number of columns: 3
## Number of rows: 100
##
## Column name: enzyme.1
## Column class: integer
##  Sample mean: 101.24
##  Sample standard deviation: 9.11
##  Sample minimum: 78
##  Sample maximum: 124
##
##  Number of missing items: 3
##
## Column name: high.credit.status
## Column class: logical
##  Proportion true: 0.81
##
##  Number of missing items: 0
##
## Column name: species
## Column class: character
##  Number of unique values: 5
##      1. Hedgehog
##      2. Aardvark
##      3. Armadillo
##
##  Number of missing items: 0
```

End of problem 1

Problem 2: Find Your Own Dataset

This problem is essentially the entire point of the course.

In this problem, we want you to find your own dataset to study and practice with.

Your dataset should be in .csv or tab-delimited text form.

Find something that interests you. As always, you'll get a lot more out of the exercise if you engage with it, and if you care about the subject matter that makes it easier for you to maintain energy.

Part (a): Dataset description

Tell us a little bit about your dataset. Write a few sentences to give us some background information about the subject. Also, tell us where you got the dataset from. When you're all done, read in the data file and store the resulting data frame in a variable.

Solution

Part (b): Number of rows

How many rows does your dataset have? Determine this using R, and report your result with one sentence.

Solution

Part (c): Number of variables

How many variables does your dataset have? Determine this using R, and report your result with one sentence.

Solution

Part (d): `NA` values

Are there any `NA` values in your dataset? Tell us with one or two sentences.

Solution

Part (e): Ask a question

Pose an interesting question about your subject matter, and then use the data in your data frame to answer this question. You should calculate some sort of numerical value such as a sample mean or a sample proportion, and report this to us. Also, you should make some sort of visualization, although the precise graph is up to you.

What's important is that the numerical result and the visualization should be relevant to answering your question.

Hint: it's probably easier to find a good quality dataset, and then think of a question that you can answer with the data, rather than to think of the question first and then try to dig up some data to answer it.

You'll get full credit as long as you do *something*, and the TAs won't grade you based on the content of your analysis. However, I think you'll get more out of this exercise if you dig in and engage with it. Once again, we come back to: find something that you care about, and then you'll find it interesting to do the analysis.

Remember: all I can do is to demonstrate the tools, and show you some basic applications. Ultimately, it's up to you to integrate this knowledge into your practice, and only you can do this.

Finally, do a nice job, and present your results in a finished manner. Amaze and delight the TAs with your imagination, creativity, precision, and attention to detail.

Solution

End of problem 2

Problem 3: Loan Amortization

In our course we usually use data frames as a way to store data. But they can also be used to construct tabular displays, and this application is sometimes underappreciated.

In this problem, we're going to use R to display all of the steps involved in constructing a loan amortization schedule.

Part (a): Payment amount

Taylor takes out a loan for $200,000, to be paid back in 10 annual installments. The interest rate for the loan is 2.8%.

Calculate the annual payment amount that Taylor needs to make in order to fully pay off this loan with 10 annual installments. Report your result using a `cat()` statement, displaying this value with 2 decimal places.

Solution

```
initial.loan.amount <- 200000

number.of.payments <- 10

interest.rate <- 0.028
```

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

```
payment.amount <-
    interest.rate * initial.loan.amount /
    (1 - discount.factor^number.of.payments)

cat(
    "Payment amount:",
    formatC(
        payment.amount,
        format = "f",
        digits = 2
    )
)
```

```
## Payment amount: 23207.42
```

Part (b): Loan amortization schedule

Construct a report that displays the loan amortization schedule for all 10 years.

To construct this report, first create three storage vectors, each of length 10:

- The first storage vector holds the current loan balance at the start of the year.
- The second storage vector holds the current loan balance at the end of the year, before the payment.
- The third storage vector holds the current loan balance at the end of the year, after the payment.

We've already seen the algorithm for constructing a loan amortization schedule.

However, for this problem, instead of printing out the values using a `cat()` statement, you should now store the result each individual calculation in one of the storage vectors:

- For year 1:
  - At the beginning of year 1, the current loan balance is just the initial loan amount. Store this in location 1 of the storage vector for the loan balance at the start of the year.
  - At the end of year 1 before the payment, the current loan balance has accrued interest. Determine the loan balance at the end of the year before the payment, and store this value in location 1 of the storage vector for the loan balance at the end of the year before the payment.
  - At the end of year 1 after the payment, the current loan balance has decreased by the payment amount. Determine this value, and store it in location 1 of the storage vector for the loan balance at the end of the year after the payment.
- Now we move on to year 2:
  - Determine the loan balance at the beginning of the year, and store this value in location 2 of the appropriate storage vector.
  - Determine the loan balance at the end of the year before the payment, and store this value in location 2 of the appropriate storage vector.
  - Determine the loan balance at the end of the year after the payment, and store this value in location 2 of the appropriate storage vector.

Continue in this way: calculate each of the three loan balances for year i, storing them in location i of the appropriate storage vector.

At the end of this calculation you will have three storage vectors, each of length 10.

When you've finished calculating all the values, combine the three storage vectors into a data frame.

Finally, display the data frame directly.

Remember that you can use any string you like as a name for the data frame columns as long as you enclose the characters in quotes.

Solution

```
loan.balance.at.start.of.year.vector <-
    numeric( number.of.payments )

loan.balance.at.end.of.year.before.payment.vector <-
    numeric( number.of.payments )

loan.balance.at.end.of.year.after.payment.vector <-
    numeric( number.of.payments )

current.balance <-
    initial.loan.amount

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

    loan.balance.at.start.of.year.vector[ year.index ] <-
        current.balance
```

```
    current.balance <-
        (1 + interest.rate) * current.balance

    loan.balance.at.end.of.year.before.payment.vector[ year.index ] <-
        current.balance

    current.balance <-
        current.balance - payment.amount

    loan.balance.at.end.of.year.after.payment.vector[ year.index ] <-
        current.balance
}
```

```
data.frame(
    "Start of year" =
        formatC(
            loan.balance.at.start.of.year.vector,
            format = "f",
            digits = 2
        ),
    "End of year, before payment" =
        formatC(
            loan.balance.at.end.of.year.before.payment.vector,
            format = "f",
            digits = 2
        ),
    "End of year, after payment" =
        formatC(
            loan.balance.at.end.of.year.after.payment.vector,
            format = "f",
            digits = 2
        )
)
```

```
##    Start.of.year End.of.year..before.payment End.of.year..after.payment
## 1      200000.00                   205600.00                  182392.58
## 2      182392.58                   187499.57                  164292.16
## 3      164292.16                   168892.34                  145684.92
## 4      145684.92                   149764.09                  126556.68
## 5      126556.68                   130100.26                  106892.84
## 6      106892.84                   109885.84                   86678.43
## 7       86678.43                    89105.42                   65898.00
## 8       65898.00                    67743.15                   44535.73
## 9       44535.73                    45782.73                   22575.31
## 10      22575.31                    23207.42                       0.00
```

End of problem 3

Problem 4: Kepler's Third Law

This is one of my favorite problems in the whole course, and I'm excited to present it to you. Kepler's laws of planetary motion were some of the most important discoveries of early modern science, and the final result is easy to visualize. But we're studying this problem in our course because it also involves many of the R techniques that we've studied so far, and so it's a great example of combining multiple methods to obtain a solution.

Before we get started, you might find it useful to go back and review Section 2 from Week 3 Module 6: Stripcharts, which gives some background for this problem.

In the year 1619, Johannes Kepler published his Third Law of Planetary Motion.

The *period* of a planet's orbit, denoted T, is the time required to make one full orbit around the sun.

By definition, the Earth has a period of 1 year.

Mars has a period of 1.88 years, while Neptune has a period of 165 years.

The mean distance from the sun is denoted by R.

Then Kepler's third law states that for any planet the square of the period T is proportional to the cube of the mean distance R:

$$T^2 \propto R^3$$

If we take the logarithm of both sides, we obtain:

$$\log T \;=\; \frac{3}{2}\log R + c$$

In other words, when we take logarithms of both sides, we have a linear relationship between the logarithm of the period of the orbit and the logarithm of the radius of the orbit. (The constant c is also very interesting, but for our purposes we'll just ignore it.)

Let X denote the logarithm of the radius:

$$X \;=\; \log R$$

Let Y denote the logarithm of the period:

$$Y \;=\; \log T$$

Then we can write Kepler's third law as:

$$Y \;=\; \frac{3}{2}X + c$$

Part (a): Read in the data

The file "Problem 8 Data.csv" contains data on the mean distance from the sun and the period for all 8 planets (sorry, Pluto).

Read this data in, and store it in a data frame variable.

Then directly display the entire data frame.

Solution

```
planets.data.frame <-
  read.csv(
      "../Problem Set 8 Data/Problem 4 Data.csv",
      stringsAsFactors = TRUE
  )

planets.data.frame
```

```
##    planet mean.distance.from.sun  period
## 1 Mercury               5.79e+10   0.241
## 2   Venus               1.08e+11   0.615
## 3   Earth               1.50e+11   1.000
## 4    Mars               2.28e+11   1.880
## 5 Jupiter               7.78e+11  11.900
## 6  Saturn               1.43e+12  29.500
## 7  Uranus               2.87e+12  84.000
## 8 Neptune               4.50e+12 165.000
```

Part (b): Log radius

Construct a vector of the logarithms of the mean radius from the sun, and store it in a variable.

Directly display the first 6 values of this vector, formatting the values with 2 decimal places.

Solution

```
log.radius.vector <-
  log( planets.data.frame$mean.distance.from.sun )

round( head( log.radius.vector ), digits = 2 )
```

```
## [1] 24.78 25.41 25.73 26.15 27.38 27.99
```

Part (c): Log period

Construct a vector of the logarithms of the periods, and store it in a variable.

Directly display the first 6 values of this vector, formatting the values with 2 decimal places.

Solution

```
log.period.vector <-
  log( planets.data.frame$period )

round( head( log.period.vector ), digits = 2 )
```

```
## [1] -1.42 -0.49  0.00  0.63  2.48  3.38
```

Part (d): Scatter plot

Construct a scatter plot of the data, using x-axis to display the logarithm of the radius and the y-axis to display the logarithm of the period.

For this scatter plot, the x-axis should range from 24 to 30, and the y-axis should range from -2 to 6.

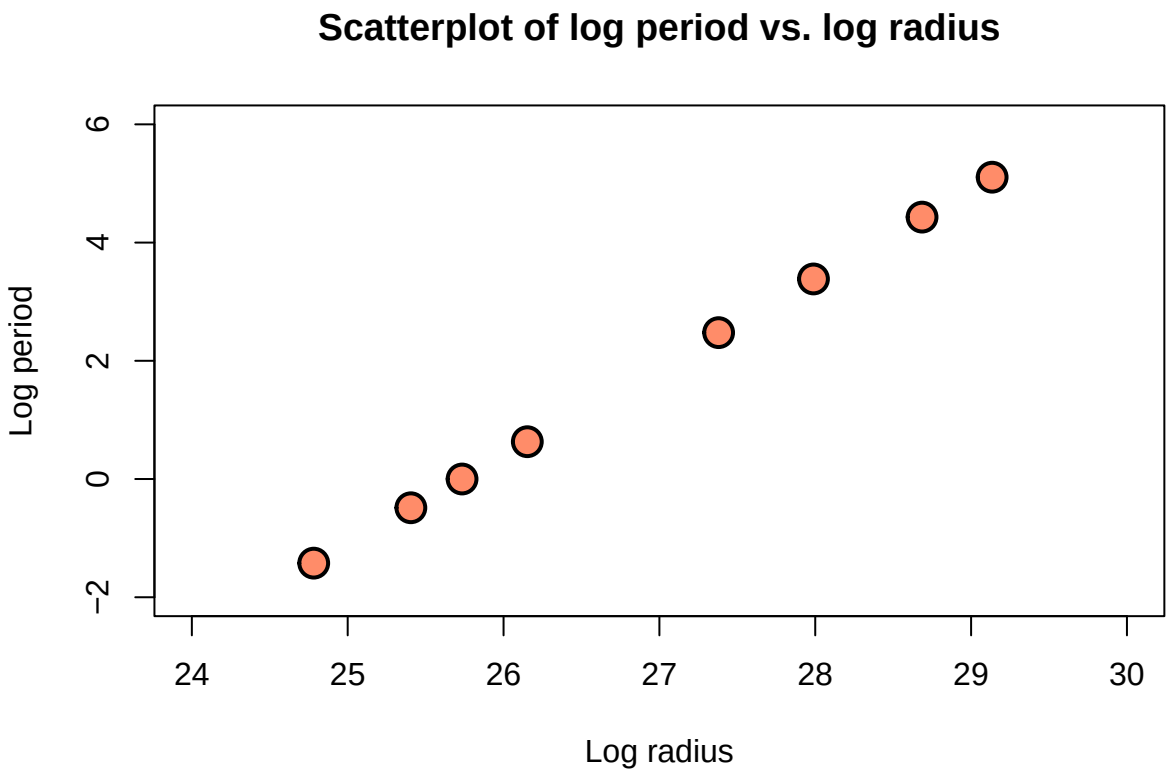
You only have 8 points for this scatter plot, so you should make them larger than normal.

Kepler's third law of planetary motion states that the relationship between the log of the radius of the orbit and the log of the period is a straight line.

Looking at your scatter plot, do you think that it supports Kepler's third law of planetary motion?

Solution

```
plot(
  x = log.radius.vector,
  y = log.period.vector,
  xlim = c(24, 30),
  ylim = c(-2, 6),
  main = "Scatterplot of log period vs. log radius",
  xlab = "Log radius",
  ylab = "Log period",
  pch = 21,
  cex = 2,
  lwd = 2,
  bg = "salmon1"
)
```



Part (e): Linear model

Kepler's third law actually says something more than the relationship between the log of the radius and the log of the period is a straight line.

It also makes a precise prediction about the slope of this straight line: it must be 3/2, or 1.5.

Construct a linear model, where the log of the radius is the predictor variable and the log of the period is the outcome variable.

You can obtain the slope of the regression line by using the `summary()` function, or by directly extracting the coefficient from the linear model object.

Determine the slope of the least-squares regression line. Is it close to 1.5?

Solution

```
linear.model <-
  lm( log.period.vector ~ log.radius.vector )

summary( linear.model )
```

```
##
## Call:
## lm(formula = log.period.vector ~ log.radius.vector)
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -0.0040977 -0.0010858  0.0000196  0.0011947  0.0034117
##
## Coefficients:
##                      Estimate Std. Error t value Pr(>|t|)
## (Intercept)       -3.860e+01  1.614e-02   -2392   <2e-16 ***
## log.radius.vector  1.500e+00  5.987e-04    2505   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.002557 on 6 degrees of freedom
## Multiple R-squared:      1,  Adjusted R-squared:      1
## F-statistic: 6.277e+06 on 1 and 6 DF,  p-value: < 2.2e-16
```

Part (f): Graphing the line

Copy your code over from part (d). Then draw the least-squares regression line from part (e) by using the `abline()` function.

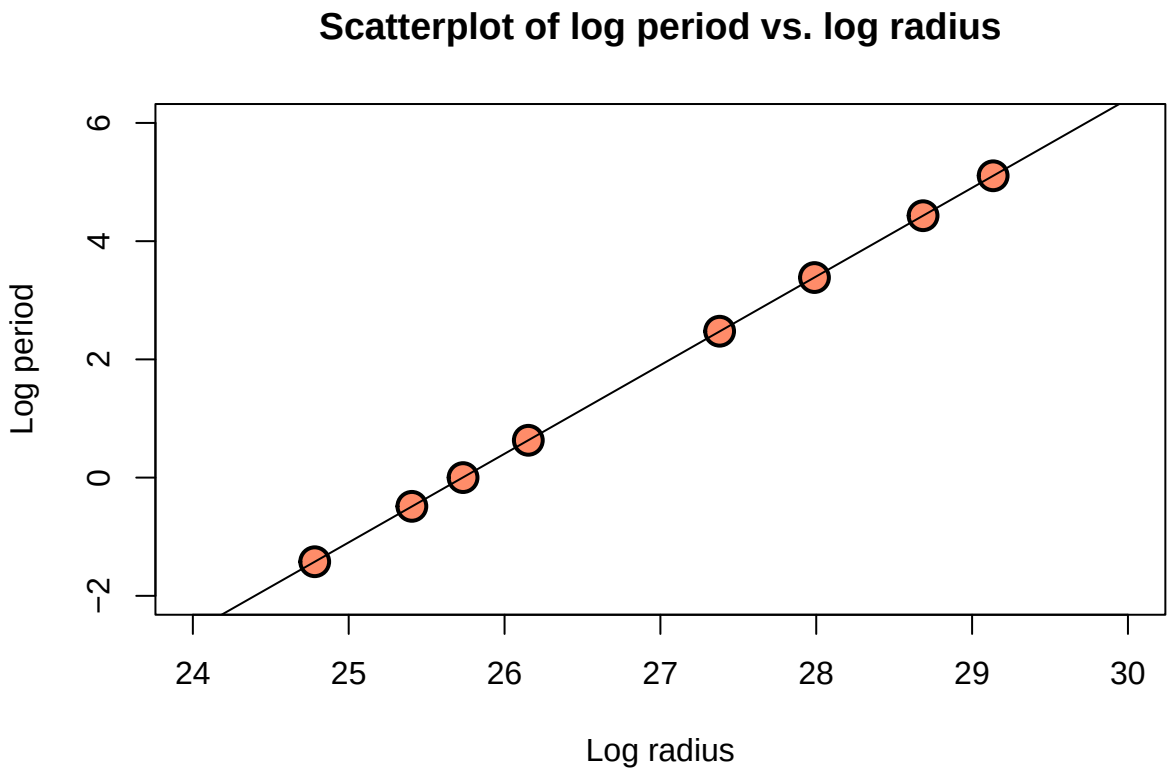
Does this graph support Kepler's third law of planetary motion?

Solution

```
plot(
  x = log.radius.vector,
  y = log.period.vector,
  xlim = c(24, 30),
  ylim = c(-2, 6),
  main = "Scatterplot of log period vs. log radius",
```

```
  xlab = "Log radius",
  ylab = "Log period",
  pch = 21,
  cex = 2,
  lwd = 2,
  bg = "salmon1"
)

abline( linear.model )
```



End of Problem 4

Problem 5: Baseball Data

Whenever we've calculated baseball batting statistics, we've always used the career totals for a player.

For instance, for Babe Ruth, we have:

Statistics	Career Total
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

However, this is not how baseball data is typically reported.

Instead, baseball statistics are typically reported on an annual basis.

That is, the data is presented for each year that a player played in the major leagues.

Babe Ruth's first season in the major leagues was 1914, and his last season was 1935, so he played a total of 22 seasons.

Thus, the data for Babe Ruth consists of 22 rows, one for each season.

The CSV file `Babe Ruth Annual Batting Data.csv` is located in the `Problem Set 8 Data` folder, in a subfolder called `Baseball Data`.

The column names for this data use these abbreviations:

Batting statistic	Abbreviation
At-Bats	AB
Hits	H
Doubles	2B
Triples	3B
Home Runs	HR
Bases on Balls	BB
Hit by a Pitch	HBP
Sacrifice Flies	SF

Part (a): Reading in the data

Read in the data from `Babe Ruth Annual Batting Data.csv` and store it in a variable. (You'll have to use a path for this.) There's nothing to report for this part, but write your code clearly so the TAs can understand what you're doing.

Solution

```
babe.ruth.annual.batting.statistics.data.frame <-
  read.csv(
    file = "../Problem Set 8 Data/Baseball Data/Babe Ruth Annual Batting Data.csv"
  )
```

Part (b): Babe Ruth career hits

Can we use the data in this format to calculate Babe Ruth's career batting average? To do this, we have to know his career total hits and his career total at-bats. In this part, we'll calculate his career total hits.

First, select the vector of hits from the data frame from part (a). Then calculate the sum of the values in this vector,and store it in a variable. Report your result using a `cat()` statement, formatting the value with 0 decimal places.

Solution

```
babe.ruth.career.total.hits <-
  sum( babe.ruth.annual.batting.statistics.data.frame$H )

cat(
  "Babe Ruth career total hits:",
  formatC(
    babe.ruth.career.total.hits,
    format = "f",
    digits = 0,
    big.mark = ","
  )
)
```

```
## Babe Ruth career total hits: 2,873
```

Part (b): Babe Ruth career total at-bats

In this part, we'll calculate Babe Ruth's career total at-bats.

First, select the vector of at-bats from the data frame from part (a). Then calculate the sum of the values in this vector,and store it in a variable. Report your result using a `cat()` statement, formatting the value with 0 decimal places.

Solution

```
babe.ruth.career.total.at.bats <-
  sum( babe.ruth.annual.batting.statistics.data.frame$AB )

cat(
  "Babe Ruth career total at-bats:",
  formatC(
    babe.ruth.career.total.at.bats,
    format = "f",
    digits = 0,
    big.mark = ","
  )
)
```

```
## Babe Ruth career total at-bats: 8,399
```

Part (d): Babe Ruth career batting average

Use your results from parts (b) and (c) to calculate Babe Ruth's career batting average. Report your result using a `cat()` statement, displaying the value using standard baseball formatting conventions.

Solution

```
babe.ruth.career.batting.average <-
  babe.ruth.career.total.hits /
  babe.ruth.career.total.at.bats

babe.ruth.career.batting.average.character.string <-
  formatC(
    babe.ruth.career.batting.average,
    format = "f",
    digits = 3
  )

babe.ruth.formatted.career.batting.average.character.string <-
  substr(
    babe.ruth.career.batting.average.character.string,
    start = 2,
    stop = 5
  )

cat(
  "Babe Ruth career batting average:",
  babe.ruth.formatted.career.batting.average.character.string
)
```

```
## Babe Ruth career batting average: .342
```

Part (e): Ted Williams career on-base percentage

A CSV file containing Ted Williams' annual baseball batting data is contained in the file named "Ted Williams Annual Baseball Data.csv", located in the `Data` folder, in a subfolder called `Baseball Data`.

Read in Ted Williams' annual baseball batting data from this file, and store it in a variable. Then use this data to calculate Ted Williams' career on-base percentage. Report your result using a `cat()` statement, displaying the value using standard baseball formatting conventions.

Note: sacrifice flies were not recorded before 1954, so we have no data on these for players who played in these years. When computing baseball statistics involving sacrifice flies for players who played during these years, the standard practice is to simply assume a value of 0 sacrifice flies for these years. You'll have to figure out how to deal with this issue in this dataset.

Solution

```
ted.williams.annual.batting.statistics.data.frame <-
  read.csv(
    file = "../Problem Set 8 Data/Baseball Data/Ted Williams Annual Batting Data.csv"
  )
```

Now let's calculate Ted Williams' career total hits:

```
ted.williams.career.total.hits <-
  sum( ted.williams.annual.batting.statistics.data.frame$H )

cat(
```

```
  "Ted Williams career total hits:",
  formatC(
    ted.williams.career.total.hits,
    format = "f",
    digits = 0,
    big.mark = ","
  )
)
```

```
## Ted Williams career total hits: 2,654
```

Now let's calculate Ted Williams' career total bases on balls:

```
ted.williams.career.total.bases.on.balls <-
  sum( ted.williams.annual.batting.statistics.data.frame$BB )

cat(
  "Ted Williams career total bases on balls:",
  formatC(
    ted.williams.career.total.bases.on.balls,
    format = "f",
    digits = 0,
    big.mark = ","
  )
)
```

```
## Ted Williams career total bases on balls: 2,021
```

Now let's calculate Ted Williams' career total number of times hit by a pitch:

```
ted.williams.career.total.hit.by.a.pitch <-
  sum( ted.williams.annual.batting.statistics.data.frame$HBP )

cat(
  "Ted Williams career total hit by pitch:",
  formatC(
    ted.williams.career.total.hit.by.a.pitch,
    format = "f",
    digits = 0,
    big.mark = ","
  )
)
```

```
## Ted Williams career total hit by pitch: 39
```

Now we can calculate Ted Williams‘ career total at-bats:

```
ted.williams.career.total.at.bats <-
  sum( ted.williams.annual.batting.statistics.data.frame$AB )

cat(
```

```
  "Ted Williams career total at-bats:",
  formatC(
    ted.williams.career.total.at.bats,
    format = "f",
    digits = 0,
    big.mark = ","
  )
)
```

```
## Ted Williams career total at-bats: 7,706
```

Now we can calculate Ted Williams‘ career total sacrifice flies:

```
ted.williams.career.total.sacrifice.flies <-
  sum(
    ted.williams.annual.batting.statistics.data.frame$SF,
    na.rm = TRUE
  )

cat(
  "Ted Williams career total sacrifice flies:",
  formatC(
    ted.williams.career.total.sacrifice.flies,
    format = "f",
    digits = 0,
    big.mark = ","
  )
)
```

```
## Ted Williams career total sacrifice flies: 20
```

Finally we can calculate Ted Williams’ career on-base percentage:

```
ted.williams.career.on.base.percentage <-
  (ted.williams.career.total.hits +
     ted.williams.career.total.bases.on.balls +
     ted.williams.career.total.hit.by.a.pitch) /
  (ted.williams.career.total.at.bats +
     ted.williams.career.total.bases.on.balls +
     ted.williams.career.total.hit.by.a.pitch +
     ted.williams.career.total.sacrifice.flies)

ted.williams.career.on.base.percentage.character.string <-
  formatC(
    ted.williams.career.on.base.percentage,
    format = "f",
    digits = 3
  )

ted.williams.career.on.base.percentage.formatted.character.string <-
  substr(
    ted.williams.career.on.base.percentage.character.string,
```

```
    start = 2,
    stop = 5
  )

cat(
  "Ted Williams career on-base percentage:",
  ted.williams.career.on.base.percentage.formatted.character.string
)
```

```
## Ted Williams career on-base percentage: .482
```

Part (f): Willie Mays career slugging percentage

A CSV file containing Willie Mays' annual baseball batting data is contained in the file named "Willie Mays Annual Baseball Data.csv", located in the `Problem Set 8 Data` folder, in a subfolder called `Baseball Data`.

Read in Willie Mays' annual baseball batting data from this file, and store it in a variable. Then use this data to calculate Willie Mays' career on-base percentage. Report your result using a `cat()` statement, displaying the value using standard baseball formatting conventions.

Solution

```
willie.mays.annual.batting.statistics.data.frame <-
  read.csv(
    file = "../Problem Set 8 Data/Baseball Data/Willie Mays Annual Batting Data.csv"
  )

names( willie.mays.annual.batting.statistics.data.frame)
```

```
##  [1] "Year"   "Age"    "Tm"     "Lg"     "G"      "PA"     "AB"     "R"
##  [9] "H"      "X2B"    "X3B"    "HR"     "RBI"    "SB"     "CS"     "BB"
## [17] "SO"     "BA"     "OBP"    "SLG"    "OPS"    "OPS."   "TB"     "GDP"
## [25] "HBP"    "SH"     "SF"     "IBB"    "Pos"    "Awards"
```

Now let's calculate Willie Mays' career total hits:

```
willie.mays.career.total.hits <-
  sum( willie.mays.annual.batting.statistics.data.frame$H )

cat(
  "Willie Mays career total hits:",
  formatC(
    willie.mays.career.total.hits,
    format = "f",
    digits = 0,
    big.mark = ","
  )
)
```

```
## Willie Mays career total hits: 3,283
```

Now let's calculate Willie Mays' career total doubles:

```
willie.mays.career.total.doubles <-
  sum( willie.mays.annual.batting.statistics.data.frame$X2B )

cat(
  "Willie Mays career total doubles:",
  formatC(
    willie.mays.career.total.doubles,
    format = "f",
    digits = 0,
    big.mark = ","
  )
)
```

```
## Willie Mays career total doubles: 523
```

Now let's calculate Willie Mays' career total triples:

```
willie.mays.career.total.triples <-
  sum( willie.mays.annual.batting.statistics.data.frame$X3B )

cat(
  "Willie Mays career total triples:",
  formatC(
    willie.mays.career.total.triples,
    format = "f",
    digits = 0,
    big.mark = ","
  )
)
```

```
## Willie Mays career total triples: 140
```

Now let's calculate Willie Mays' career total home runs:

```
willie.mays.career.total.home.runs <-
  sum( willie.mays.annual.batting.statistics.data.frame$HR )

cat(
  "Willie Mays career total home runs:",
  formatC(
    willie.mays.career.total.home.runs,
    format = "f",
    digits = 0,
    big.mark = ","
  )
)
```

```
## Willie Mays career total home runs: 660
```

Now let's calculate Willie Mays' career total singles:

```
willie.mays.career.total.singles <-
  willie.mays.career.total.hits -
  (willie.mays.career.total.doubles +
     willie.mays.career.total.triples +
     willie.mays.career.total.home.runs)
```

Now let's calculate Willie Mays' career total at-bats:

```
willie.mays.career.total.at.bats <-
  sum( willie.mays.annual.batting.statistics.data.frame$AB )

cat(
  "Willie Mays career total at-bats:",
  formatC(
    willie.mays.career.total.at.bats,
    format = "f",
    digits = 0,
    big.mark = ","
  )
)
```

```
## Willie Mays career total at-bats: 10,881
```

Now we can calculate Willie Mays' career total bases:

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

Now we can calculate Willie Mays‘ career slugging percentage:

```
willie.mays.career.slugging.percentage <-
  willie.mays.career.total.bases /
  willie.mays.career.total.at.bats

willie.mays.career.slugging.percentage.character.string <-
  formatC(
    willie.mays.career.slugging.percentage,
    format = "f",
    digits = 3
  )

willie.mays.career.slugging.percentage.formatted.character.string <-
  substr(
    willie.mays.career.slugging.percentage.character.string,
    start = 2,
    stop = 5
  )

cat(
```

```
  "Willie Mays career slugging percentage:",
  willie.mays.career.slugging.percentage.formatted.character.string
)
```

```
## Willie Mays career slugging percentage: .557
```

End of Problem 5

Problem 6: Final Grades, Encore

You might have noticed that we still haven't fully solved the problem of calculating final grades for our course.

The remaining issue is that I've always told you what the total problem set scores are, but in reality this is the sum of the 11 individual problem set scores.

The file `Problem 6 Data.csv` contains data for a class of 80 students:

- Registration status
- The final raw score for each of the 11 problem sets
- The raw score for the Midterm Assessment
- The raw score for the Comprehensive Assessment

Use the grading system we discussed in Week 2 Module 4: Your Final Grade. However, for this problem you should use this grading schedule:

Range	Letter Grade
90 <= Score	A
80 <= Score < 90	B
70 <= Score < 80	C
60 <= Score < 70	D
Score < 60	F

Your challenge in this problem is to calculate the final letter grade for each of the students of this class.

When you're all done, directly display a relative proportions table of the letter grades, and then use this table to construct a barplot of the letter grade frequency counts. Remember to include all the required elements of the barplot.

You're on your own for this one! You'll have to decide how to implement this, especially how to obtain the total problem set raw scores.

Solution

```
problem.6.data.frame <-
  read.csv(
    file = "../Problem Set 8 Data/Problem 6 Data.csv"
  )

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

```
##               Name Registration.status Problem.Set.0 Problem.Set.1
## 1         Amy Chen    (08046) Graduate             8             5
## 2       Alex James    (82569) Graduate             8             3
## 3       John Bayes    (65162) Graduate             8             6
## 4     Brian Wilson    (96169) Graduate             8             6
## 5 Matthew Thompson    (41437) Graduate             8             6
## 6      Julie Bayes    (30766) Graduate             8             5
##   Problem.Set.2 Problem.Set.3 Problem.Set.4 Problem.Set.5 Problem.Set.6
## 1             4             6             6             3             2
```

```
## 2             5             4             5             5             5
## 3             6             5             4             5             4
## 4             5             6             6             6             6
## 5             6             5             5             5             5
## 6             4             5             3             5             5
##   Midterm.Assessment Problem.Set.7 Problem.Set.8 Problem.Set.9 Problem.Set.10
## 1                 66             4             4             3              4
## 2                 77             5             6             5              4
## 3                 77             5             5             5              5
## 4                 72             6             6             6              6
## 5                 55             3             6             6              6
## 6                 71             5             5             4              2
##   Comprehensive.Assessment
## 1                       55
## 2                       59
## 3                       66
## 4                       53
## 5                       56
## 6                       71
```

```
registration.status.vector <-
  substr(
    x = problem.6.data.frame$Registration.status,
    start = 9,
    stop = 13
  )

head( registration.status.vector, n = 8 )
```

```
## [1] "Gradu" "Gradu" "Gradu" "Gradu" "Gradu" "Gradu" "Gradu" "Gradu"
```

Here's one approach:

```
problem.set.raw.score.vector <-
  problem.6.data.frame$Problem.Set.0 +
  problem.6.data.frame$Problem.Set.1 +
  problem.6.data.frame$Problem.Set.2 +
  problem.6.data.frame$Problem.Set.3 +
  problem.6.data.frame$Problem.Set.4 +
  problem.6.data.frame$Problem.Set.5 +
  problem.6.data.frame$Problem.Set.6 +
  problem.6.data.frame$Problem.Set.7 +
  problem.6.data.frame$Problem.Set.8 +
  problem.6.data.frame$Problem.Set.9 +
  problem.6.data.frame$Problem.Set.10

problem.set.raw.score.vector
```

```
##  [1] 49 55 58 67 61 51 62 54 67 68 57 54 63 63 58 48 52 50 68 57 63 60 65 52 52
## [26] 63 56 60 59 54 68 59 62 67 51 65 56 52 58 52 50 66 68 62 49 62 62 66 64 49
## [51] 50 62 58 62 49 55 63 51 51 57 50 60 63 50 57 63 66 57 66 66 58 63 61 54 59
## [76] 53 55 54 51 63
```

Here's another approach:

```
problem.set.raw.score.data.frame <-
    problem.6.data.frame[
        c( "Problem.Set.0", "Problem.Set.1", "Problem.Set.2",
           "Problem.Set.3", "Problem.Set.4", "Problem.Set.5",
           "Problem.Set.6", "Problem.Set.7", "Problem.Set.8",
           "Problem.Set.9", "Problem.Set.10" )
    ]

problem.set.raw.score.vector <-
    rowSums( problem.set.raw.score.data.frame )

problem.set.raw.score.vector
```

```
##  [1] 49 55 58 67 61 51 62 54 67 68 57 54 63 63 58 48 52 50 68 57 63 60 65 52 52
## [26] 63 56 60 59 54 68 59 62 67 51 65 56 52 58 52 50 66 68 62 49 62 62 66 64 49
## [51] 50 62 58 62 49 55 63 51 51 57 50 60 63 50 57 63 66 57 66 66 58 63 61 54 59
## [76] 53 55 54 51 63
```

```
midterm.assessment.raw.score.vector <-
  problem.6.data.frame$Midterm.Assessment

midterm.assessment.raw.score.vector
```

```
##  [1] 66 77 77 72 55 71 55 66 65 70 56 63 80 71 62 63 63 76 74 68 65 78 74 76 78
## [26] 62 79 72 57 72 67 72 75 65 77 56 74 56 72 62 59 71 75 70 63 69 76 65 62 65
## [51] 67 80 66 56 74 55 76 67 58 64 61 66 67 61 80 73 58 64 71 60 80 66 68 80 80
## [76] 65 63 62 55 65
```

```
comprehensive.assessment.raw.score.vector <-
  problem.6.data.frame$Comprehensive.Assessment

comprehensive.assessment.raw.score.vector
```

```
##  [1] 55 59 66 53 56 71 67 78 79 63 76 76 59 60 55 67 67 66 74 50 68 72 68 77 74
## [26] 61 62 59 61 68 60 61 51 58 59 77 53 65 64 53 60 50 67 50 60 66 62 67 57 54
## [51] 68 72 61 51 62 67 68 50 70 77 77 76 51 78 56 64 73 59 80 59 73 79 57 52 54
## [76] 70 59 61 64 78
```

```
problem.set.standardized.score.vector <-
  problem.set.raw.score.vector / 68 * 100
```

```
midterm.assessment.standardized.score.vector <-
  midterm.assessment.raw.score.vector / 80 * 100
```

```
comprehensive.assessment.standardized.score.vector <-
  comprehensive.assessment.raw.score.vector / 80 * 100
```

```r
preliminary.score.1.vector <-
  (0.2 * problem.set.standardized.score.vector) +
  (0.3 * midterm.assessment.standardized.score.vector) +
  (0.5 * comprehensive.assessment.standardized.score.vector)
```

```r
preliminary.score.2.vector <-
  (0.35 * midterm.assessment.standardized.score.vector) +
  (0.65 * comprehensive.assessment.standardized.score.vector)
```

```r
graduate.course.score.vector <-
  ifelse(
    preliminary.score.1.vector > preliminary.score.2.vector,
    preliminary.score.1.vector,
    preliminary.score.2.vector
  )
```

```r
final.course.score.vector <-
  ifelse(
    registration.status.vector == "Under",
    4/3 * graduate.course.score.vector,
    graduate.course.score.vector
  )

final.course.score.vector
```

```
##   [1]  73.56250  81.92647  87.31250  79.83088  73.56618  88.75000  80.73529
##   [8]  92.25000  93.45588  85.62500  86.25000  89.31250  85.40441  82.65441
##  [15]  74.68382  82.00000  82.00000  86.87500  94.00000  73.51471  85.40441
##  [22] 123.50000 119.15686 127.75000  94.25000  79.90441 113.25000  81.52206
##  [29]  76.85294  86.75000  82.62500  82.47794  78.23529  80.33088 108.83333
##  [36]  88.24265  77.34559  77.31250  84.05882  71.66912  74.56250  77.28676
##  [43] 120.00000  75.73529  76.31250  85.36029  85.48529  85.66176  77.69853
##  [50]  72.53676  84.56250  93.50000  79.93382  71.11029  82.75000  78.67647
##  [57]  89.52941  71.37500  82.25000  90.56250 119.00000  90.62500  75.52941
##  [64]  90.06250  81.76471  85.90441  86.78676 103.51961  96.06250  78.78676
##  [71]  94.31250  93.06250  79.06618  78.38235  81.10294  85.31250  76.67647
##  [78]  77.25735  76.06250  91.81250
```

```r
letter.grades.vector <-
  cut(
    final.course.score.vector,
    breaks = c(-Inf, 60, 70, 80, 90, Inf),
    labels = c( "F", "D", "C", "B", "A" ),
    right = FALSE
  )
```

```r
letter.grades.relative.proportions.table <-
  proportions(
      table( letter.grades.vector )
  )

letter.grades.relative.proportions.table
```

```
## letter.grades.vector
##    F    D    C    B    A
## 0.00 0.00 0.35 0.40 0.25
```

```
barplot(
  letter.grades.relative.proportions.table,
  ylim = c(0, 0.5),
  main = "Barplot of final letter grades",
  xlab = "Letter grades",
  ylab = "Frequency count",
  col = "skyblue3"
)
```