Assignment3

Anthony Conrardy

2024-02-06

Exercise #1-FiveThirtyEight College Major Data

The data was retrieved directly from the FiveThirtyEight website GitHub. It should be noted that the data was last committed over ten (10) years ago.

```
data_538_majors <- read.csv("https://raw.githubusercontent.com/fivethirtyeight/data/master/colle
ge-majors/majors-list.csv")
```

Identifying Data or Statistics Majors

Identify all majors that contain either “DATA” or “STATISTICS”

```
#Identifies whether an element has the word "DATA" and places in data1 (regardless of case).
data1 <- grep(pattern = 'data', data_538_majors$Major, value = TRUE, ignore.case = TRUE)
#Identifies whether an element has the word "STATISTICS" and places in statistics (regardless of
case).
statistics <- grep(pattern = 'stati', data_538_majors$Major, value = TRUE, ignore.case = TRUE)
# Combines the identified statistics majors.
statscombined <- paste(statistics, collapse = ", ")
#Combines data and statistics majors.
dads <- paste(data1,statscombined, sep= ", ")
#Reports result
dads
```

```
## [1] "COMPUTER PROGRAMMING AND DATA PROCESSING, MANAGEMENT INFORMATION SYSTEMS AND STATISTICS,
STATISTICS AND DECISION SCIENCE"
```

Exercise #2-Data Transformation

Given a list of data, transform the string data into format like:

c(“bell pepper”, “bilberry”, “blackberry”, “blood orange”, “blueberry”, “cantaloupe”, “chili pepper”, “cloudberry”, “elderberry”, “lime”, “lychee”, “mulberry”, “olive”, “salal berry”)

Admittedly, I may have taken this to be a little too literal.

```
berry_data <- read.csv("https://raw.githubusercontent.com/Aconrard/DATA607/main/Assignment3/berr
y_data_assignment3.csv", header = FALSE)
berry_name <- berry_data$V1
#Extract first 13 elements.
bn1 <- berry_name[1:13]
#Extract 14th element.
bn2 <- berry_name[14]
#Add quotation marks and comma, collapse the strings of first 13 elements.
bn3 <- str_c('"',bn1,'", ', collapse = "")
#Add quotation marks and collapse the string of the 14th element.
bn4 <- str_c('"',bn2,'"', collapse = "")
#Add all 14 elements together, add "c(", and collapse string.
bn5 <- str_c('c(',bn3,bn4, collapse = "")
#Add final ")" and collapse string.
bn6 <- str_c(bn5,')',collapse = "")
cat(bn6)
```

```
## c("bell pepper", "bilberry", "blackberry", "blood orange", "blueberry", "cantaloupe", "chili
pepper", "cloud berry", "elderberry", "lime", "lychee", "mulberry", "olive", "salal berry")
```

Exercise #3-Pattern Expressions

This is a very complex topic though it appears to be very useful for data exploration. The methods for pattern recognition are sometimes difficult to follow, but I will makes some assumptions on the front end to make this as useful as possible:

1. It will be assumed that all the expressions were encased in the proper quotes. Without the proper quotes, the search terms would not work correctly within the functions.
2. On page 267 of the textbook (the one in my possession), the authors state they write expressions in a limited format without the extra escapes and quotes, so an expression of (.)\1\1 will assumed to be "(.)\\1\\1".

Describe in words, what these expressions will match:

a. (.)\1\1 or "(.)\\1\\1" In this case the (.) matches any single character and places it in a group. The \1"back references" the character in the group, and the second \1 also "back references" the character in the group. So the expression should match patterns like 'aaa'.

b. "(.)(.)\\2\\1" In this case, the first (.) matches any single character and places it into a group. The second (.) matches any single character and places it into a second group. The \\2 back references the character in the second group, and \\1 back references the character in the first group. So the expression should match patterns like 'abba'.

c. (..)\1 or "(..)\\1" In this case, the (..) matches any two characters and places it into a group. The \\1 back references the two characters in the group. So the expression should match patterns like "abab" or "aaaa".

d. "(.).\1.\1" This case gets a little more complex. The (.) matches any single character and places it into a group. The second '.' matches any character. The \1 back references the character contained in the group, while the second '.' matches any character. The final \1 again back references the character contained in the goup. So the expression should match patterns like 'abaca', 'aaaaa', 'abaaa', or 'aaaba'.

e. "(.)(.)(.).\\3\\2\\*1" In this final case, each (.) matches any single character and places it into a group. The .* matches any number of characters including"0". The order of the groups created by each (.) will determine the back references. Therefore, \\3 back references the third (.), the \\2 back references the second (.), and the \\1 back references the first (.). So the expression should match patterns like "abc9876cba", or where the first three characters are mirrored at the end of the pattern.

Exercise #4-Construction of Matching Expressions

For the purposes of this exercise, I created a CSV files of strings that will be used to check the accuracy of the pattern match. There are 100 character strings, 42 characters in length.

```
dstr <- read.csv("strings.csv")
tibble(dstr)
```

```
## # A tibble: 100 × 1
##    string_names
##    <chr>
##  1 ERJWYDINBCNEWNXGZNOSDDOPOCFUSPEKOTSDQANAMW
##  2 QWQAEAEASBRBKLPNIRGGGKALIYORMLJVXOZXLAAOYL
##  3 AIZPQMTDGNCSSAZRGYZIKBIZCVNJWMBWUTZJKPVSVW
##  4 PRIWJMOCNINSFVZHKOPGYMPPXVPZKTRDGHPEWNAIJO
##  5 QXCCXUFKWXWYBGUWMIJTZKGQLHFKWYHSEUFJZFMBCT
##  6 PRFJWUAWFTHQJLYWAGQSKAZKEFKQXZOPZEKEJTOTPR
##  7 NMIXLBSORUATJGDHJVEESZCVZHOQOHYVPLNTVHWJQQ
##  8 TWAMFHJSYRDHBGZSGOHUQEIFAVAZGRKIAERLREMOOE
##  9 ESBGVJSGYAXDLCFHSTIHGRMFVXLRCRLPMKLMVZWZBA
## 10 AABBOFWSLRYYOHQHPXVLINYYXVKXYKDLYLAHXARBFZ
## # ℹ 90 more rows
```

Construct regular expressions to match words that:

a. Start and end with the same character

   "(.).*\\1$"

   The (.) picks any first character

   The .* allows any length of intervening characters

   \1 back references the character that is grouped in (.)

   $ identifies that it occurs at the end of the string.

```
grep(pattern = "(.).*\\1$", dstr$string_names, value = TRUE, ignore.case = TRUE)
```

```
## [1] "EEBDBQMGQRLIXUFASNUZMUXIDQNVUTMGLZYXWNIBUE"
## [2] "PUYGCWXFPBCIMSAJFXIAYVCXIWCBBTAKEFGQBDQWLP"
## [3] "SFASQWMPASAIDFUDDWNGWAZIERQACQKOWFGAWLCHHS"
## [4] "GZVLOKFLOTLKQPAFQDTQLUBRYKUZUVXHAZBUBRYJRG"
```

b. Contain a repeated pair of letters (e.g. "church" contains "ch" repeated twice.)

“(..).*\\1”

(..) matches any two word character and places in first group.

.* allows for any length of charcaters between

\\1 back references the initial two characters, too make it twice in a string.

```
str_view(dstr$string_names, "(..).*\\1")
```

```
##  [1] │ ERJWYDINBCNEWNXGZNO<SDDOPOCFUSPEKOTSD>QANAMW
##  [2] │ QWQ<AEAE>ASBRBKLPNIRGGGKALIYORMLJVXOZXLAAOYL
##  [3] │ A<IZPQMTDGNCSSAZRGYZIKBIZ>CVNJWMBWUTZJKPVSVW
##  [5] │ QXCCX<UFKWXWYBGUWMIJTZKGQLHFKWYHSEUF>JZFMBCT
##  [6] │ <PRFJWUAWFTHQJLYWAGQSKAZKEFKQXZOPZEKEJTOTPR>
## [10] │ AABBOFWSLR<YYOHQHPXVLINYY>XVKXYKDLYLAHXARBFZ
## [11] │ UZ<ZNGLJZN>AWWAJRZPOIWHPKXLUMGSGVAKYSZVRXZLI
## [12] │ LPELFCK<WWXWHMDLUNBTPCRYXWW>VVKSFSHTKHDFIKONU
## [14] │ ZQQA<IHKZUVWRGODDPKWDGRVQXJEDGYZLKTZZYZIH>XL
## [15] │ KXWOBKFDKQ<ADBZVMVIIVCLMOVCCEVABGYYBAD>HOMPE
## [16] │ W<SISHPRGLVWOVUPNJHJNLEBZCFMSI>UTCSDXTUHTFQH
## [17] │ EEBDBQ<MGQRLIXUFASNUZMUXIDQNVUTMG>LZYXWNIBUE
## [18] │ FEYOUZRBZQFNJS<OMFPGUQTCVDPEOM>UAZSUCMROYRAN
## [20] │ UUENMCONTNZQGADTQ<QMTSKYPDLPYSJYMUXMXFPQM>FV
## [22] │ PUYGCWXFPBCIMSAJF<XIAYVCXI>WCBBTAKEFGQBDQWLP
## [23] │ I<XUCFDHMWYWAXU>GJHLLZGOEZKDNARGFWUFNBXDMCHX
## [25] │ MTZLVNCSONYXREXPHIK<BTXAFBT>BUGWTFTMWNAJATAQ
## [26] │ TBVFPGSQEQORAGTJRFDWLTMA<TAVUXYROGDSTA>HFUZN
## [27] │ QXV<ADATFNIZVYFOFFNYJGQTTNDPJXLHZKAD>IFMBTTS
## [28] │ N<NUQJIJLCLEYEDUTIEUXNU>HRHTUNZSICKGQWKMHARO
## ... and 51 more
```

c. Contain one letter repeated in at least three places (e.g. “eleven” contains three “e”s.)

“(\\w).* \\1.* \\1”

(\\w) matches any word character in the group.

.* allows for any length of charcaters between

\\1 back references the initial two characters, to make it twice in a string.

.* allows for any length of charcaters between

\\1 back references the initial two characters, to make it three in a string.

```
str_view(dstr$string_names, "(\\w).*\\1.*\\1")
```

```
##  [1] │ <ERJWYDINBCNEWNXGZNOSDDOPOCFUSPE>KOTSDQANAMW
##  [2] │ QWQ<AEAEASBRBKLPNIRGGGKALIYORMLJVXOZXLAA>OYL
##  [3] │ A<IZPQMTDGNCSSAZRGYZIKBI>ZC<VNJWMBWUTZJKPVSV>W
##  [4] │ <PRIWJMOCNINSFVZHKOPGYMPPXVPZKTRDGHP>EWNAIJO
##  [5] │ Q<XCCXUFKWX><WYBGUWMIJTZKGQLHFKW>YHSEUFJZFMBCT
##  [6] │ <PRFJWUAWFTHQJLYWAGQSKAZKEFKQXZOPZEKEJTOTP>R
##  [7] │ NMIXLBS<ORUATJGDHJVEESZCVZHOQO>HYVPLNTVHWJQQ
##  [8] │ TW<AMFHJSYRDHBGZSGOHUQEIFAVAZGRKIA><ERLREMOOE>
##  [9] │ E<SBGVJSGYAXDLCFHS>TIHG<RMFVXLRCR>LPMKLMVZWZBA
## [10] │ <AABBOFWSLRYYOHQHPXVLINYYXVKXYKDLYLAHXA>RBFZ
## [11] │ U<ZZNGLJZNAWWAJRZPOIWHPKXLUMGSGVAKYSZVRXZ>LI
## [12] │ <LPELFCKWWXWHMDL>UNBTPCRYXWWVV<KSFSHTKHDFIK>ONU
## [13] │ W<QVKSFHSIPXNKVFGOFQWAJEUMZVRAVUEGRTUZTQ>BIM
## [14] │ <ZQQAIHKZUVWRGODDPKWDGRVQXJEDGYZLKTZZYZ>IHXL
## [15] │ <KXWOBKFDK>Q<ADBZVMVIIVCLMOVCCEVABGYYBA>DHOMPE
## [16] │ W<SISHPRGLVWOVUPNJHJNLEBZCFMSIUTCS>DXTUHTFQH
## [17] │ <EEBDBQMGQRLIXUFASNUZMUXIDQNVUTMGLZYXWNIBUE>
## [18] │ <FEYOUZRBZQFNJSOMF>PG<UQTCVDPEOMUAZSU>CMROYRAN
## [19] │ J<POPGKALCNGETEXBZDP>JO<FYXLEERWDFBUWBVF>ZTWZG
## [20] │ <UUENMCONTNZQGADTQQMTSKYPDLPYSJYMU>XMXFPQMFV
## ... and 80 more
```

Exercise #4 Note

It needs to be noted that some of the identified strings seem to have more than three (3) identified same characters. That is due to overlapping pattern identification of the str_view expression. In [2] you can easily see five “A”s in the string. That is because we are seeing the following identification by the expression:

<AAA>AA

A<AAA>A

AA<AAA>