post02-John-Nipp

John Nipp

November 28, 2017

Analyzing Text with Strings in R

Introduction:

Common knowledge is that R is not a scripting language, and as such, does not have the string manipulation capabilities that other scripting languages have. The truth is, common knowledge is right. However, string manipulation is still important in data processing.

We need to understand how reformat strings, combine strings, create subsets of strings, and print strings, and use regular expressions in order to properly use R's full capabilities for computing data. In this post, we will review base functions, functions available in the stringr package and functions available in the stringi package. We will then end with an example that utilizes all of these functions.

My Motivation for this Topic:

The reason why I chose to write on this topic is because of a recent assignment in manipulation of strings (lab11). I felt like understanding strings would help simplify some processes in data cleaning or help when taking input in a Shiny App. I also believed this was a dense topic, and there weren't a lot of comprehensive resources on the subject that organized the information well. So my goal in this post is to give a rudimentary introduction to this topic that combines all of the infomration in a digestible way.

I chose to leave out certain functions that are of popular use. I just want you, the reader, to walk away having a clear idea of how to get information from text.

We will use the following built in R data tables for this topic (from the datasets package: - fruit - Seatbelts

```
library(stringr) # Will use this later.
```

```
## Warning: package 'stringr' was built under R version 3.4.2
```

```
library(datasets)
fruit # List of fruit
```

```
##  [1] "apple"             "apricot"           "avocado"          
##  [4] "banana"            "bell pepper"       "bilberry"         
##  [7] "blackberry"        "blackcurrant"      "blood orange"     
## [10] "blueberry"         "boysenberry"       "breadfruit"       
## [13] "canary melon"      "cantaloupe"        "cherimoya"        
## [16] "cherry"            "chili pepper"      "clementine"       
## [19] "cloudberry"        "coconut"           "cranberry"        
## [22] "cucumber"          "currant"           "damson"           
## [25] "date"              "dragonfruit"       "durian"           
## [28] "eggplant"          "elderberry"        "feijoa"           
## [31] "fig"               "goji berry"        "gooseberry"       
## [34] "grape"             "grapefruit"        "guava"            
## [37] "honeydew"          "huckleberry"       "jackfruit"        
## [40] "jambul"            "jujube"            "kiwi fruit"       
## [43] "kumquat"           "lemon"             "lime"             
## [46] "loquat"            "lychee"            "mandarine"        
## [49] "mango"             "mulberry"          "nectarine"        
## [52] "nut"               "olive"             "orange"           
## [55] "pamelo"            "papaya"            "passionfruit"     
## [58] "peach"             "pear"              "persimmon"        
## [61] "physalis"          "pineapple"         "plum"             
## [64] "pomegranate"       "pomelo"            "purple mangosteen"
## [67] "quince"            "raisin"            "rambutan"         
## [70] "raspberry"         "redcurrant"        "rock melon"       
## [73] "salal berry"       "satsuma"           "star fruit"       
## [76] "strawberry"        "tamarillo"         "tangerine"        
## [79] "ugli fruit"        "watermelon"
```

```
Seatbelts # Data on car accidents
```

```
##          DriversKilled drivers front rear   kms PetrolPrice VanKilled law
## Jan 1969           107    1687   867  269  9059  0.10297181        12   0
## Feb 1969            97    1508   825  265  7685  0.10236300         6   0
## Mar 1969           102    1507   806  319  9963  0.10206249        12   0
## Apr 1969            87    1385   814  407 10955  0.10087330         8   0
## May 1969           119    1632   991  454 11823  0.10101967        10   0
## Jun 1969           106    1511   945  427 12391  0.10058119        13   0
## Jul 1969           110    1559  1004  522 13460  0.10377398        11   0
## Aug 1969           106    1630  1091  536 14055  0.10407640         6   0
## Sep 1969           107    1579   958  405 12106  0.10377398        10   0
## Oct 1969           134    1653   850  437 11372  0.10302640        16   0
## Nov 1969           147    2152  1109  434  9834  0.10273011        13   0
## Dec 1969           180    2148  1113  437  9267  0.10199719        14   0
## Jan 1970           125    1752   925  316  9130  0.10127456        14   0
## Feb 1970           134    1765   903  311  8933  0.10070398         6   0
## Mar 1970           110    1717  1006  351 11000  0.10013961         8   0
```

```
## Apr 1970          102    1558   892   362 10733  0.09862110        11   0
## May 1970          103    1575   990   486 12912  0.09834929         7   0
## Jun 1970          111    1520   866   429 12926  0.09808018        13   0
## Jul 1970          120    1805  1095   551 13990  0.09727921        13   0
## Aug 1970          129    1800  1204   646 14926  0.09741062        11   0
## Sep 1970          122    1719  1029   456 12900  0.09742524        11   0
## Oct 1970          183    2008  1147   475 12034  0.09638063        14   0
## Nov 1970          169    2242  1171   456 10643  0.09573896        16   0
## Dec 1970          190    2478  1299   468 10742  0.09510631        14   0
## Jan 1971          134    2030   944   356 10266  0.09673597        17   0
## Feb 1971          108    1655   874   271 10281  0.09610922        16   0
## Mar 1971          104    1693   840   354 11527  0.09536725        15   0
## Apr 1971          117    1623   893   427 12281  0.09470959        13   0
## May 1971          157    1805  1007   465 13587  0.09411762        13   0
## Jun 1971          148    1746   973   440 13049  0.09353215        15   0
## Jul 1971          130    1795  1097   539 16055  0.09295405        12   0
## Aug 1971          140    1926  1194   646 15220  0.09283979         6   0
## Sep 1971          136    1619   988   457 13824  0.09272474         9   0
## Oct 1971          140    1992  1077   446 12729  0.09226965        13   0
## Nov 1971          187    2233  1045   402 11467  0.09170669        14   0
## Dec 1971          150    2192  1115   441 11351  0.09126207        15   0
## Jan 1972          159    2080  1005   359 10803  0.09071160        14   0
## Feb 1972          143    1768   857   334 10548  0.09027633         3   0
## Mar 1972          114    1835   879   312 12368  0.08995192        12   0
## Apr 1972          127    1569   887   427 13311  0.08909964        13   0
## May 1972          159    1976  1075   434 13885  0.08867919        12   0
## Jun 1972          156    1853  1121   486 14088  0.08815929         8   0
## Jul 1972          138    1965  1190   569 16932  0.08890206         8   0
## Aug 1972          120    1689  1058   523 16164  0.08818133        15   0
## Sep 1972          117    1778   939   418 14883  0.08894029         8   0
## Oct 1972          170    1976  1074   452 13532  0.08772661         5   0
## Nov 1972          168    2397  1089   462 12220  0.08742885        17   0
## Dec 1972          198    2654  1208   497 12025  0.08703543        14   0
## Jan 1973          144    2097   903   354 11692  0.08644992        13   0
## Feb 1973          146    1963   916   347 11081  0.08587264         5   0
## Mar 1973          109    1677   787   276 13745  0.08539822         8   0
## Apr 1973          131    1941  1114   472 14382  0.08382198         5   0
## May 1973          151    2003  1014   487 14391  0.08459078        12   0
## Jun 1973          140    1813  1022   505 15597  0.08413690        11   0
## Jul 1973          153    2012  1114   619 16834  0.08377841        13   0
## Aug 1973          140    1912  1132   640 17282  0.08351074        15   0
## Sep 1973          161    2084  1111   559 15779  0.08280639        11   0
## Oct 1973          168    2080  1008   453 13946  0.08117889        11   0
## Nov 1973          152    2118   916   418 12701  0.08285361        10   0
## Dec 1973          136    2150   992   419 10431  0.09419012        13   0
## Jan 1974          113    1608   731   262 11616  0.09239984         8   0
## Feb 1974          100    1503   665   299 10808  0.10816148         6   0
## Mar 1974          103    1548   724   303 12421  0.10721169         8   0
## Apr 1974          103    1382   744   401 13605  0.11404297        14   0
## May 1974          121    1731   910   413 14455  0.11245412        12   0
## Jun 1974          134    1798   883   426 15019  0.11131625        14   0
## Jul 1974          133    1779   900   516 15662  0.11030125        13   0
## Aug 1974          129    1887  1057   600 16745  0.10819718         9   0
## Sep 1974          144    2004  1076   459 14717  0.10702744         4   0
## Oct 1974          154    2077   919   443 13756  0.10494698        13   0
## Nov 1974          156    2092   920   412 12531  0.11935775         6   0
## Dec 1974          163    2051   953   400 12568  0.11762190        15   0
## Jan 1975          122    1577   664   278 11249  0.13302742        12   0
## Feb 1975           92    1356   607   302 11096  0.13084524        16   0
## Mar 1975          117    1652   777   381 12637  0.12831848         7   0
## Apr 1975           95    1382   633   279 13018  0.12354745        12   0
## May 1975           96    1519   791   442 15005  0.11858681        10   0
## Jun 1975          108    1421   790   409 15235  0.11633748         9   0
## Jul 1975          108    1442   803   416 15552  0.11516148         9   0
## Aug 1975          106    1543   884   511 16905  0.11450120         6   0
## Sep 1975          140    1656   769   393 14776  0.11352298         7   0
## Oct 1975          114    1561   732   345 14104  0.11193018        13   0
## Nov 1975          158    1905   859   391 12854  0.11061053        14   0
## Dec 1975          161    2199   994   470 12956  0.11527439        13   0
## Jan 1976          102    1473   704   266 12177  0.11379349        14   0
## Feb 1976          127    1655   684   312 11918  0.11234958        11   0
## Mar 1976          125    1407   671   300 13517  0.11175347        11   0
## Apr 1976          101    1395   643   373 14417  0.10964252        10   0
## May 1976           97    1530   771   412 15911  0.10844090         4   0
## Jun 1976          112    1309   644   322 15589  0.10788494         8   0
## Jul 1976          112    1526   828   458 16543  0.10908477         9   0
## Aug 1976          113    1327   748   427 17925  0.10757145        10   0
## Sep 1976          108    1627   767   346 15406  0.10616402        10   0
## Oct 1976          128    1748   825   421 14601  0.10630000         5   0
## Nov 1976          154    1958   810   344 13107  0.10482531        13   0
## Dec 1976          162    2274   986   370 12268  0.10345175        12   0
## Jan 1977          112    1648   714   291 11972  0.10144992        10   0
## Feb 1977           79    1401   567   224 12028  0.10040232         9   0
## Mar 1977           82    1411   616   266 14033  0.09886203         7   0
## Apr 1977          127    1403   678   338 14244  0.10249615         5   0
```

```
## May 1977          108    1394   742   298 15287  0.10302743        10    0
## Jun 1977          110    1520   840   386 16954  0.10217891         5    0
## Jul 1977          123    1528   888   479 17361  0.09983664         6    0
## Aug 1977          103    1643   852   473 17694  0.09263669         8    0
## Sep 1977           97    1515   774   332 16222  0.09181496         6    0
## Oct 1977          140    1685   831   391 14969  0.09072430        12    0
## Nov 1977          165    2000   889   370 13624  0.09002121        15    0
## Dec 1977          183    2215  1046   431 13842  0.08933071         7    0
## Jan 1978          148    1956   889   366 12387  0.08844273        14    0
## Feb 1978          111    1462   626   250 11608  0.08835257         4    0
## Mar 1978          116    1563   808   355 15021  0.08675736        10    0
## Apr 1978          115    1459   746   304 14834  0.08499524         8    0
## May 1978          100    1446   754   379 16565  0.08456794         7    0
## Jun 1978          106    1622   865   440 16882  0.08443190        11    0
## Jul 1978          134    1657   980   500 18012  0.08435088         3    0
## Aug 1978          125    1638   959   511 18855  0.08360098         5    0
## Sep 1978          117    1643   856   384 17243  0.08341726        11    0
## Oct 1978          122    1683   798   366 16045  0.08274514        10    0
## Nov 1978          153    2050   942   432 14745  0.08523527        10    0
## Dec 1978          178    2262  1010   390 13726  0.08477030         7    0
## Jan 1979          114    1813   796   306 11196  0.08445892        10    0
## Feb 1979           94    1445   643   232 12105  0.08535212        11    0
## Mar 1979          128    1762   794   342 14723  0.08755921         9    0
## Apr 1979          119    1461   750   329 15582  0.09038292         7    0
## May 1979          111    1556   809   394 16863  0.09078329         8    0
## Jun 1979          110    1431   716   355 16758  0.10874278        13    0
## Jul 1979          114    1427   851   385 17434  0.11414223         8    0
## Aug 1979          118    1554   931   463 18359  0.11299293         5    0
## Sep 1979          115    1645   834   453 17189  0.11132071         8    0
## Oct 1979          132    1653   762   373 16909  0.10912623         7    0
## Nov 1979          153    2016   880   401 15380  0.10769846        12    0
## Dec 1979          171    2207  1077   466 15161  0.10760157        10    0
## Jan 1980          115    1665   748   306 14027  0.10377502         7    0
## Feb 1980           95    1361   593   263 14478  0.10711417         4    0
## Mar 1980           92    1506   720   323 16155  0.10737477        10    0
## Apr 1980          100    1360   646   310 16585  0.11169537         4    0
## May 1980           95    1453   765   424 18117  0.11063818         8    0
## Jun 1980          114    1522   820   403 17552  0.11185521         8    0
## Jul 1980          102    1460   807   406 18299  0.10974234         7    0
## Aug 1980          104    1552   885   466 19361  0.10819393        10    0
## Sep 1980          132    1548   803   381 17924  0.10625536         8    0
## Oct 1980          136    1827   860   369 17872  0.10419303        14    0
## Nov 1980          117    1737   825   378 16058  0.10193397         8    0
## Dec 1980          137    1941   911   392 15746  0.10279382         9    0
## Jan 1981          111    1474   704   284 15226  0.10476034         8    0
## Feb 1981          106    1458   691   316 14932  0.10400254         6    0
## Mar 1981           98    1542   688   321 16846  0.11665552         7    0
## Apr 1981           84    1404   714   358 16854  0.11516148         6    0
## May 1981           94    1522   814   378 18146  0.11298954         5    0
## Jun 1981          105    1385   736   382 17559  0.11386064         4    0
## Jul 1981          123    1641   876   433 18655  0.11911808         5    0
## Aug 1981          109    1510   829   506 19453  0.12448999        10    0
## Sep 1981          130    1681   818   428 17923  0.12322295         7    0
## Oct 1981          153    1938   942   479 17915  0.12067793        10    0
## Nov 1981          134    1868   782   370 16496  0.12104898        12    0
## Dec 1981           99    1726   823   349 13544  0.11696857         7    0
## Jan 1982          115    1456   595   238 13601  0.11275026         4    0
## Feb 1982          104    1445   673   285 15667  0.10807931         5    0
## Mar 1982          131    1456   660   324 17358  0.10883852         6    0
## Apr 1982          108    1365   676   346 18112  0.11129177         4    0
## May 1982          103    1487   755   410 18581  0.11130401         4    0
## Jun 1982          115    1558   815   411 18759  0.11545436         8    0
## Jul 1982          122    1488   867   496 20668  0.11476830         8    0
## Aug 1982          122    1684   933   534 21040  0.11720743         3    0
## Sep 1982          125    1594   798   396 18993  0.11907640         7    0
## Oct 1982          137    1850   950   470 18668  0.11796586        12    0
## Nov 1982          138    1998   825   385 16768  0.11744913         2    0
## Dec 1982          152    2079   911   411 16551  0.11698846         7    0
## Jan 1983          120    1494   619   281 16231  0.11261054         8    0
## Feb 1983           95    1057   426   300 15511  0.11365702         3    1
## Mar 1983          100    1218   475   318 18308  0.11314445         2    1
## Apr 1983           89    1168   556   391 17793  0.11849553         6    1
## May 1983           82    1236   559   398 19205  0.11796940         3    1
## Jun 1983           89    1076   483   337 19162  0.11768661         7    1
## Jul 1983           60    1174   587   477 20997  0.12005924         6    1
## Aug 1983           84    1139   615   422 20705  0.11943775         8    1
## Sep 1983          113    1427   618   495 18759  0.11888127         8    1
## Oct 1983          126    1487   662   471 19240  0.11846236         4    1
## Nov 1983          122    1483   519   368 17504  0.11801660         3    1
## Dec 1983          118    1513   585   345 16591  0.11770662         5    1
## Jan 1984           92    1357   483   296 16224  0.11777609         5    1
## Feb 1984           86    1165   434   319 16670  0.11479699         3    1
## Mar 1984           81    1282   513   349 18539  0.11573525         4    1
## Apr 1984           84    1110   548   375 19759  0.11535626         3    1
## May 1984           87    1297   586   441 19584  0.11481536         6    1
```

```
## Jun 1984          90    1185   522  465 19976  0.11477748         6   1
## Jul 1984          79    1222   601  472 21486  0.11493598         7   1
## Aug 1984          96    1284   644  521 21626  0.11479699         5   1
## Sep 1984         122    1444   643  429 20195  0.11409316         7   1
## Oct 1984         120    1575   641  408 19928  0.11646552         7   1
## Nov 1984         137    1737   711  490 18564  0.11602611         4   1
## Dec 1984         154    1763   721  491 18149  0.11606673         7   1
```

Strings

What are strings?

In R, strings are sequences of characters expressed within a pair of single or double quotes.

```
"This is a string" # Double quotes
'This is also a string' # Single quotes
This is not a string # No quotes!
```

```
## Error: <text>:3:6: unexpected symbol
## 2: 'This is also a string' # Single quotes
## 3: This is
##          ^
```

In order to use quotations within a string, one should either use quotations that aren't used to contain the string, OR, use a backslash before. That way one is working with the interpreter properly.

```
"This 'example' is fine" # Singles within doubles
"This \"example\" is also fine" # Using backslashes before doubles (within doubles)
'So is this "example"' # Doubles within singles
"But this "not so good example"doesn't work" # Doubles within doubles with no backslashes
```

```
## Error: <text>:4:12: unexpected symbol
## 3: 'So is this "example"' # Doubles within singles
## 4: "But this "not
##               ^
```

How make something into a string?

Here are two popular functions to use:

The function toString is a base function in R. It takes any R object, coerces it to character, and then separates each element with "," to create a single string. Very nice function to create strings using vectors.

```
toString(fruit) # Using character vector fruit
```

```
## [1] "apple, apricot, avocado, banana, bell pepper, bilberry, blackberry, blackcurrant, blood orange, blueberry,
boysenberry, breadfruit, canary melon, cantaloupe, cherimoya, cherry, chili pepper, clementine, cloudberry, coconu
t, cranberry, cucumber, currant, damson, date, dragonfruit, durian, eggplant, elderberry, feijoa, fig, goji berry,
gooseberry, grape, grapefruit, guava, honeydew, huckleberry, jackfruit, jambul, jujube, kiwi fruit, kumquat, lemon
, lime, loquat, lychee, mandarine, mango, mulberry, nectarine, nut, olive, orange, pamelo, papaya, passionfruit, p
each, pear, persimmon, physalis, pineapple, plum, pomegranate, pomelo, purple mangosteen, quince, raisin, rambutan
, raspberry, redcurrant, rock melon, salal berry, satsuma, star fruit, strawberry, tamarillo, tangerine, ugli frui
t, watermelon"
```

If one is using a data.frame, then toString will combine each column at the end of the preceding one in order into a single string.

```
substr(toString(Seatbelts), 1, 1000)  # Seatbelts (has numerics) first 1000
```

```
## [1] "107, 97, 102, 87, 119, 106, 110, 106, 107, 134, 147, 180, 125, 134, 110, 102, 103, 111, 120, 129, 122, 183
, 169, 190, 134, 108, 104, 117, 157, 148, 130, 140, 136, 140, 187, 150, 159, 143, 114, 127, 159, 156, 138, 120, 11
7, 170, 168, 198, 144, 146, 109, 131, 151, 140, 153, 140, 161, 168, 152, 136, 113, 100, 103, 103, 121, 134, 133, 1
29, 144, 154, 156, 163, 122, 92, 117, 95, 96, 108, 108, 106, 140, 114, 158, 161, 102, 127, 125, 101, 97, 112, 112,
113, 108, 128, 154, 162, 112, 79, 82, 127, 108, 110, 123, 103, 97, 140, 165, 183, 148, 111, 116, 115, 100, 106, 13
4, 125, 117, 122, 153, 178, 114, 94, 128, 119, 111, 110, 114, 118, 115, 132, 153, 171, 115, 95, 92, 100, 95, 114,
102, 104, 132, 136, 117, 137, 111, 106, 98, 84, 94, 105, 123, 109, 130, 153, 134, 99, 115, 104, 131, 108, 103, 115
, 122, 122, 125, 137, 138, 152, 120, 95, 100, 89, 82, 89, 60, 84, 113, 126, 122, 118, 92, 86, 81, 84, 87, 90, 79,
96, 122, 120, 137, 154, 1687, 1508, 1507, 1385, 1632, 1511, 1559, 1630, 1579, 1653, 2152, 2148,"
```

```
                                        # characters, substr() is a later topic
```

The width parameter can be used to specify the max length of the resulting string.

```
toString(fruit, width = 10) # Fruit character vector, with first 10 characters
```

```
## [1] "apple,...."
```

```
                              # of each elemeent
```

An alternative you could use is as.character(), but it won't manipulate the text in the same manner if you are using a dataframe, vector, or list.

Each element will remain separate, but every element will become a character type.

```
as.character(Seatbelts[,1]) # Coerces first column to character
```

```
##   [1] "107" "97"  "102" "87"  "119" "106" "110" "106" "107" "134" "147"
##  [12] "180" "125" "134" "110" "102" "103" "111" "120" "129" "122" "183"
##  [23] "169" "190" "134" "108" "104" "117" "157" "148" "130" "140" "136"
##  [34] "140" "187" "150" "159" "143" "114" "127" "159" "156" "138" "120"
##  [45] "117" "170" "168" "198" "144" "146" "109" "131" "151" "140" "153"
##  [56] "140" "161" "168" "152" "136" "113" "100" "103" "103" "121" "134"
##  [67] "133" "129" "144" "154" "156" "163" "122" "92"  "117" "95"  "96"
##  [78] "108" "108" "106" "140" "114" "158" "161" "102" "127" "125" "101"
##  [89] "97"  "112" "112" "113" "108" "128" "154" "162" "112" "79"  "82"
## [100] "127" "108" "110" "123" "103" "97"  "140" "165" "183" "148" "111"
## [111] "116" "115" "100" "106" "134" "125" "117" "122" "153" "178" "114"
## [122] "94"  "128" "119" "111" "110" "114" "118" "115" "132" "153" "171"
## [133] "115" "95"  "92"  "100" "95"  "114" "102" "104" "132" "136" "117"
## [144] "137" "111" "106" "98"  "84"  "94"  "105" "123" "109" "130" "153"
## [155] "134" "99"  "115" "104" "131" "108" "103" "115" "122" "122" "125"
## [166] "137" "138" "152" "120" "95"  "100" "89"  "82"  "89"  "60"  "84"
## [177] "113" "126" "122" "118" "92"  "86"  "81"  "84"  "87"  "90"  "79"
## [188] "96"  "122" "120" "137" "154"
```

```
as.character(c(1,2,3,4)) # Coercing numeric vector to character
```

```
## [1] "1" "2" "3" "4"
```

```
as.character(Seatbelts)[1:200] # Coercing Seatbelts to a character vector
```

```
##   [1] "107"  "97"   "102"  "87"   "119"  "106"  "110"  "106"  "107"  "134"
##  [11] "147"  "180"  "125"  "134"  "110"  "102"  "103"  "111"  "120"  "129"
##  [21] "122"  "183"  "169"  "190"  "134"  "108"  "104"  "117"  "157"  "148"
##  [31] "130"  "140"  "136"  "140"  "187"  "150"  "159"  "143"  "114"  "127"
##  [41] "159"  "156"  "138"  "120"  "117"  "170"  "168"  "198"  "144"  "146"
##  [51] "109"  "131"  "151"  "140"  "153"  "140"  "161"  "168"  "152"  "136"
##  [61] "113"  "100"  "103"  "103"  "121"  "134"  "133"  "129"  "144"  "154"
##  [71] "156"  "163"  "122"  "92"   "117"  "95"   "96"   "108"  "108"  "106"
##  [81] "140"  "114"  "158"  "161"  "102"  "127"  "125"  "101"  "97"   "112"
##  [91] "112"  "113"  "108"  "128"  "154"  "162"  "112"  "79"   "82"   "127"
## [101] "108"  "110"  "123"  "103"  "97"   "140"  "165"  "183"  "148"  "111"
## [111] "116"  "115"  "100"  "106"  "134"  "125"  "117"  "122"  "153"  "178"
## [121] "114"  "94"   "128"  "119"  "111"  "110"  "114"  "118"  "115"  "132"
## [131] "153"  "171"  "115"  "95"   "92"   "100"  "95"   "114"  "102"  "104"
## [141] "132"  "136"  "117"  "137"  "111"  "106"  "98"   "84"   "94"   "105"
## [151] "123"  "109"  "130"  "153"  "134"  "99"   "115"  "104"  "131"  "108"
## [161] "103"  "115"  "122"  "122"  "125"  "137"  "138"  "152"  "120"  "95"
## [171] "100"  "89"   "82"   "89"   "60"   "84"   "113"  "126"  "122"  "118"
## [181] "92"   "86"   "81"   "84"   "87"   "90"   "79"   "96"   "122"  "120"
## [191] "137"  "154"  "1687" "1508" "1507" "1385" "1632" "1511" "1559" "1630"
```

```
                               # (first 200 elements)
```

Text Processing

> Text processing is about extracting useful information from text, which includes basic steps of pre-processing data, stemming the data, … and obtaining the associations between terms. R provides several libraries and functions to efficiently carry out these tasks.

- Taken from opensourceforu.com

Base Functions

Base functions are provided directly from R syntax.

Before we process text, we need to import it into global environment. We can do this with readLines(), a function that takes in a file or url, and outputs a character where elements are separated by lines in the file or from the webpage. Unnlike a function like read.csv, this function makes no assumption as to how the data is formatted. # Make this G

```
# Importing ROmeo and Juliet line for line
Romeo_Juliet <- readLines(
 "http://www.textfiles.com/etext/AUTHORS/SHAKESPEARE/shakespeare-romeo-48.txt")
```

Changing Case

If you want the case in your characters to be uniform (especially important when comparing wordcounts), any of the following three will take a character vector, string OR an object that can be coerced by as.character into a string with the specified case. * tolower() * toupper() * casefold() - upper = TRUE/FALSE is the additional parameter to decide on upper or lowe

```
tolower("TRASH")
```

```
## [1] "trash"
```

```
toupper("TRAsh")
```

```
## [1] "TRASH"
```

```
tolower("Trash")
```

```
## [1] "trash"
```

```
casefold(fruit[1:20], upper = TRUE)
```

```
##  [1] "APPLE"        "APRICOT"      "AVOCADO"      "BANANA"
##  [5] "BELL PEPPER"  "BILBERRY"     "BLACKBERRY"   "BLACKCURRANT"
##  [9] "BLOOD ORANGE" "BLUEBERRY"    "BOYSENBERRY"  "BREADFRUIT"
## [13] "CANARY MELON" "CANTALOUPE"   "CHERIMOYA"    "CHERRY"
## [17] "CHILI PEPPER" "CLEMENTINE"   "CLOUDBERRY"   "COCONUT"
```

```
casefold(c(seq(1,20)))
```

```
##  [1] "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"  "9"  "10" "11" "12" "13" "14"
## [15] "15" "16" "17" "18" "19" "20"
```

Paste

One of the most significnat functions to know is paste(). Professor Sanchez has remarked many times on how important of a function it is.

I think of function paste() as a much better version of toString. With paste, we can: - input more than one R object (which is coerced to character type) - specify what separates strings that are in the input - if we want to take one list/vector of strings, we can use collapse to make them all into one, where collapse specifies the separation between elements - Note: when we use two vectors in paste, the recycling effect occurs until all elements of all vectors have been used. I

```
paste("Money", "is", "good") # Simple
```

```
## [1] "Money is good"
```

```
paste("Money", "is", "good", sep = "? Uhh... ") # Patrick Star voice
```

```
## [1] "Money? Uhh... is? Uhh... good"
```

```
expensive_vector <- c("Money", "is", "good") # Can I get a discount?
paste(expensive_vector) # Vector must be specified to collapse
```

```
## [1] "Money" "is"    "good"
```

```
paste(expensive_vector, collapse = " (you broke my vector you moron...) ")
```

```
## [1] "Money (you broke my vector you moron...) is (you broke my vector you moron...) good"
```

```
cheap_vector <- c("lovers", "a", "book, you", "grubbers") # Creativity
paste(expensive_vector, cheap_vector) # Recycling rule at play
```

```
## [1] "Money lovers"    "is a"            "good book, you" "Money grubbers"
```

```
paste(expensive_vector, cheap_vector, collapse = " ") # Does recycling
```

```
## [1] "Money lovers is a good book, you Money grubbers"
```

```
              # and then collapses
```

The function paste0() is paste() that always separates with an empty character.

```
paste0(c(1,2)) # Simple
```

```
## [1] "1" "2"
```

```
paste0("Charlie", "and", "the", "chocolate", "factory") # Run-on sentence
```

```
## [1] "Charlieandthechocolatefactory"
```

```r
paste0(c("Charlie", "and", "the", "chocolate", "factory"), collapse = " ")
```

```
## [1] "Charlie and the chocolate factory"
```

```r
#collapsing is still similar
```

Split em up, Split em up

Don’t forget it

What do you think strsplit() does? You are probably right! strsplit takes a character vector, and splits it into smaller elements, where each element of the original vector now is now a character vector within a larger list. The Perl parameter set to true enables us to use regular expressions, but this is a later topic. An important note is that strsplit deletes the split element.

```r
strsplit(fruit[1:10], split = "") # Split at every character
```

```
## [[1]]
## [1] "a" "p" "p" "l" "e"
##
## [[2]]
## [1] "a" "p" "r" "i" "c" "o" "t"
##
## [[3]]
## [1] "a" "v" "o" "c" "a" "d" "o"
##
## [[4]]
## [1] "b" "a" "n" "a" "n" "a"
##
## [[5]]
##  [1] "b" "e" "l" "l" " " "p" "e" "p" "p" "e" "r"
##
## [[6]]
## [1] "b" "i" "l" "b" "e" "r" "r" "y"
##
## [[7]]
##  [1] "b" "l" "a" "c" "k" "b" "e" "r" "r" "y"
##
## [[8]]
##  [1] "b" "l" "a" "c" "k" "c" "u" "r" "r" "a" "n" "t"
##
## [[9]]
##  [1] "b" "l" "o" "o" "d" " " "o" "r" "a" "n" "g" "e"
##
## [[10]]
## [1] "b" "l" "u" "e" "b" "e" "r" "r" "y"
```

```r
strsplit(fruit[1:10], split = "[ae]", perl = FALSE) # Split at [ae]
```

```
## [[1]]
## [1] ""    "ppl"
##
## [[2]]
## [1] ""       "pricot"
##
## [[3]]
## [1] ""    "voc" "do"
##
## [[4]]
## [1] "b" "n" "n"
##
## [[5]]
## [1] "b"    "ll p" "pp"   "r"
##
## [[6]]
## [1] "bilb" "rry"
##
## [[7]]
## [1] "bl"  "ckb" "rry"
##
## [[8]]
## [1] "bl"     "ckcurr" "nt"
##
## [[9]]
## [1] "blood or" "ng"
##
## [[10]]
## [1] "blu" "b"   "rry"
```

```r
strsplit(fruit[1:10], split = "[ae]", perl = TRUE) # Split at a or e (later)
```

```
## [[1]]
## [1] ""    "ppl"
##
## [[2]]
## [1] ""       "pricot"
##
## [[3]]
## [1] ""    "voc" "do"
##
## [[4]]
## [1] "b" "n" "n"
##
## [[5]]
## [1] "b"    "ll p" "pp"   "r"
##
## [[6]]
## [1] "bilb" "rry"
##
## [[7]]
## [1] "bl"  "ckb" "rry"
##
## [[8]]
## [1] "bl"     "ckcurr" "nt"
##
## [[9]]
## [1] "blood or" "ng"
##
## [[10]]
## [1] "blu" "b"   "rry"
```

This can be kind of annoying, since we might not want a list of character vectors. So we use the function unlist(), a function that returns the vector we want.

```r
Tser <- strsplit(c("you tease too much", "please stop", "hurts my feelings"),
                 split = " ") #
Tser # List format
```

```
## [[1]]
## [1] "you"   "tease" "too"   "much"
##
## [[2]]
## [1] "please" "stop"
##
## [[3]]
## [1] "hurts"    "my"       "feelings"
```

```r
unlist(Tser) # Much better for manipulating later
```

```
## [1] "you"      "tease"    "too"      "much"     "please"   "stop"
## [7] "hurts"    "my"       "feelings"
```

SubStrings

A substring is a subset of the character sequence forming an entire string. In other words, it's part of a string.

A basic function is substr(). It takes a character string or character vector, a starting index per string, and an ending index per string. It handles vectors in a similar way to paste.

```r
chr_vc <- c("This", "is", "a", "character", "vector")
paste(chr_vc) # See how it's the same
```

```
## [1] "This"      "is"        "a"         "character" "vector"
```

```r
substr(chr_vc, 1, 3) # Modifying each element within the character vector
```

```
## [1] "Thi" "is"  "a"   "cha" "vec"
```

```r
substr(chr_vc, 2, 4) # First one took indecies 1 through 3, now 2 through 4
```

```
## [1] "his" "s"   ""    "har" "ect"
```

Sometimes, we find substrings in order to replace them with something else. Luckily, there is a function called sub(). First parameter is what characters you are looking to replace, second is a replacement. The last parameter is a vector/string you want to modify. There is also a Perl parameter for regular expressions (later).

```
chr_vc
```

```
## [1] "This"      "is"        "a"         "character" "vector"
```

```
sub(c("i","e"), "o", chr_vc) #See how this doesn't work?
```

```
## Warning in sub(c("i", "e"), "o", chr_vc): argument 'pattern' has length > 1
## and only the first element will be used
```

```
## [1] "Thos"      "os"        "a"         "character" "vector"
```

```
sub("i", "o", chr_vc) # Replace i with o
```

```
## [1] "Thos"      "os"        "a"         "character" "vector"
```

```
sub("a", "u", chr_vc) # Replace a with u
```

```
## [1] "This"      "is"        "u"         "churacter" "vector"
```

```
sub("[aeiou]", "y", chr_vc, perl = TRUE) # Using a regular expression
```

```
## [1] "Thys"      "ys"        "y"         "chyracter" "vyctor"
```

```
                                          # to replace all vowels with y
```

What if we want to access the indecies of a character vector that have a certain pattern? The function grep() is very useful for this! (Perl argument exists). If you set value = true, you'll get a character vector (save you a step)!

```
frt <- grep("o", fruit[1:40]) # Assigning indecies of fruit with letter o to
                              # frt (first 40 elements)
grep("i", fruit[1:40]) # Indecies of fruit with letter i (first 40 elements)
```

```
##  [1]  2  6 12 15 17 18 26 27 30 31 32 35 39
```

```
grep("a", fruit[1:40]) # Indecies of fruit with letter a (first 40 elements)
```

```
##  [1]  1  2  3  4  7  8  9 12 13 14 15 21 23 24 25 26 27 28 30 34 35 36 39
## [24] 40
```

```
fruit[frt] # Compare this
```

```
##  [1] "apricot"      "avocado"      "blood orange" "boysenberry"
##  [5] "canary melon" "cantaloupe"   "cherimoya"    "cloudberry"
##  [9] "coconut"      "damson"       "dragonfruit"  "feijoa"
## [13] "goji berry"   "gooseberry"   "honeydew"
```

```
grep("o", fruit[1:40], value = TRUE) # to this!
```

```
##  [1] "apricot"      "avocado"      "blood orange" "boysenberry"
##  [5] "canary melon" "cantaloupe"   "cherimoya"    "cloudberry"
##  [9] "coconut"      "damson"       "dragonfruit"  "feijoa"
## [13] "goji berry"   "gooseberry"   "honeydew"
```

Sets of strings

If you have taken set theory, this next part is going to be very easy for you. These functions have to do with comparing two vectors and the contents within them.

- union() - Takes two sets and creates a new set that takes all elements from both sets and deletes repeats.
- intersection() - Takes two sets and creates a new set that only contains elements thta are in both sets (repeats are deleted).
- setdiff() - Returns a vector of elements in the first set and not in the second.
- setequal() - Returns whether two vectors have all of the same elements (regardless of repeats).
- is.element() - is the first input within the second input (vector)
- sort() arranges elements in alphabetical order, decreasing = TRUE means opposite alphabetical order

```r
# first 20 fruit that also have the letter o
first20_o <- grep("o", fruit[1:20], value = TRUE)

# first 20 fruit that also have the letter i
first20_i <- grep("i", fruit[1:20], value = TRUE)

# first 20 fruit that also have the letter i
# and five "apricot" elements
new25_i <- first20_i
new25_i[7:11] <- rep("apricot", 5)


union(first20_i, first20_o) # All elements of first 20 fruit with i or o
```

```
##  [1] "apricot"      "bilberry"     "breadfruit"   "cherimoya"
##  [5] "chili pepper" "clementine"   "avocado"      "blood orange"
##  [9] "boysenberry"  "canary melon" "cantaloupe"   "cloudberry"
## [13] "coconut"
```

```r
intersect(first20_i, first20_o) # Elements that have an i or o in them
```

```
## [1] "apricot"   "cherimoya"
```

```r
setdiff(first20_i, first20_o) # Elements with i in them but not o
```

```
## [1] "bilberry"     "breadfruit"   "chili pepper" "clementine"
```

```r
setdiff(first20_o, first20_i) # Elements with o in them but not i
```

```
## [1] "avocado"      "blood orange" "boysenberry"  "canary melon"
## [5] "cantaloupe"   "cloudberry"   "coconut"
```

```r
# Do the two sets have all the same unique elements?
setequal(first20_i, first20_o)
```

```
## [1] FALSE
```

```r
# Do the two sets have all the same unique elements?
setequal(first20_i, first20_i)
```

```
## [1] TRUE
```

```r
# Do the two sets have all the same unique elements?
setequal(new25_i, first20_i)
```

```
## [1] TRUE
```

```r
is.element("apricot", first20_i) # Is apricot in set first20_i?
```

```
## [1] TRUE
```

```r
is.element("meth", first20_i) # Is meth in set first20_i
```

```
## [1] FALSE
```

```r
# Sort first 20 elemnts in opposite alphabetical order
sort(first20_i, decreasing = TRUE)
```

```
## [1] "clementine"   "chili pepper" "cherimoya"    "breadfruit"
## [5] "bilberry"     "apricot"
```

```r
sort(c("b", "a", "d", "c")) # Sort them in alphabetical order
```

```
## [1] "a" "b" "c" "d"
```

Display it baby

The function print() is very common within coding languages. It displays the results within the console. Can take a variety of objects. If you set quote to false you will have a display without quotes. The function noquote() has this set to false already.

```r
print("Charlieeee") # Prints "Charlieeee"
```

```
## [1] "Charlieeee"
```

```r
print(fruit[1:20]) # Prints the first 20 elements in fruit
```

```
##  [1] "apple"        "apricot"      "avocado"      "banana"
##  [5] "bell pepper"  "bilberry"     "blackberry"   "blackcurrant"
##  [9] "blood orange" "blueberry"    "boysenberry"  "breadfruit"
## [13] "canary melon" "cantaloupe"   "cherimoya"    "cherry"
## [17] "chili pepper" "clementine"   "cloudberry"   "coconut"
```

```r
print(fruit[1:10], quote = FALSE) # Prints the first 20 elements in fruit
```

```
##  [1] apple        apricot      avocado      banana       bell pepper
##  [6] bilberry     blackberry   blackcurrant blood orange blueberry
```

```r
                              # with no quotes.
noquote(fruit[1:10]) # Prints first 10 elements with no quotes
```

```
##  [1] apple        apricot      avocado      banana       bell pepper
##  [6] bilberry     blackberry   blackcurrant blood orange blueberry
```

The function cat() fill can be used to specify the width of output displayed, sep is the same as in paste, file can tell you where to put the file, append determines whether output is appended to the file or not. This function is extremely useful for exporting text into other files. The Apend attribute determines But if you don't have a file location, it'll display everything on the console.

```r
cat("dog", " ", sep = "\n") #Display dog
```

```
## dog
##
```

```r
cat("My grocery list: \n", fruit[1:20], sep = "... ") # Fruitatarian
```

```
## My grocery list:
## ... apple... apricot... avocado... banana... bell pepper... bilberry... blackberry... blackcurrant... blood ora
nge... blueberry... boysenberry... breadfruit... canary melon... cantaloupe... cherimoya... cherry... chili pepper
... clementine... cloudberry... coconut
```

```r
cat('\n', 0, 1, 2, 3, 4, sep = '\n') # Display numerics
```

```
##
##
## 0
## 1
## 2
## 3
## 4
```

```r
cat(fruit[1:3], sep = " heh yeah that sounds really good\n") # Hungry?
```

```
## apple heh yeah that sounds really good
## apricot heh yeah that sounds really good
## avocado
```

```r
cat("Dre", sep = "\n", file = "Dre.txt") # Write Dre in a test file Dre.txt
                                         # within the directory
                                         # (create if need)

# Add Dre and  a sequences of 0s to the file Dre.txt
cat("Dre", c(0,0,0), fruit, sep = "\n", file = "Dre.txt", append = TRUE)
```

StringR

A lot of people like the StringR package, since it has a lot of functions similar to the ones in base R, except they are better. This is partially because the stringr functions will autodetect which arguments go where. They also keep the same data stucture for output as input and they deal with NAs and empty characters well.

Generalities

These functions are very similar to functions in base R, so I will leave out the examples. - str_c() - Similar to paste() (same parameters). - str_to_upper - Just like toupper with an additional parameter (locale). - str_to_lower - Just like tolower, also has locale. - str_split - Very similar to strsplit. Input and then a pattern. - str_sub() - In addition to functionality of sub(), extracts negative indecies (0 cannot be within the interval)

If you want to know the number of characters within a string, you can use str_length(). It takes a string, a vector of strings, or some object that can be coerced into character and returns a number.

```
str_length(c("as;dlfj", "sldjfsdlk")) # How long is each element?
```

```
## [1] 7 9
```

```
str_length("this might be kind of long") #How ong is the string?
```

```
## [1] 26
```

```
fruit[1:5]
```

```
## [1] "apple"        "apricot"      "avocado"      "banana"       "bell pepper"
```

```
str_length(fruit[1:5]) # How long are the first 5 elements of fruit
```

```
## [1]  5  7  7  6 11
```

```
str_length(121) # Coerces to character then returns length of it
```

```
## [1] 3
```

The function str_extract() extracts patterns in the string(s), otherwise returns an NA. The function str_extract_all() extracts as many instances as possible, but returns a list… unless you set simplify to TRUE.

```
str_extract("eseses", pattern = "s") # Returns first s
```

```
## [1] "s"
```

```
str_extract_all("eseses", pattern = "s") # Returns every s
```

```
## [[1]]
## [1] "s" "s" "s"
```

```
# Returns every s in a matrix, one row per string
str_extract_all(c("eseses", "ses"), pattern = "s", simplify = TRUE)
```

```
##      [,1] [,2] [,3]
## [1,] "s"  "s"  "s" 
## [2,] "s"  "s"  ""
```

```
# Returns all five first fruit with occurences of letter e in one list per elem
str_extract(fruit[1:5], pattern = "e")
```

```
## [1] "e" NA  NA  NA  "e"
```

```
# Returns all five first fruit with all
# occurences of letter e in one list per elem
str_extract_all(fruit[1:5], pattern = "e")
```

```
## [[1]]
## [1] "e"
## 
## [[2]]
## character(0)
## 
## [[3]]
## character(0)
## 
## [[4]]
## character(0)
## 
## [[5]]
## [1] "e" "e" "e"
```

The functions str_replace() and str_replace_all() are similar to the extract functions, except it's one or more replacements. Its a substitution, like sub().

```
# Replace first o in chocolate mouse with a (boston accent)
str_replace("chocolate mouse", pattern = "o", replacement = "a")
```

```
## [1] "chacolate mouse"
```

```r
# Replace all substrings in the first five elements of  fruit with
# abdominable fruit
str_replace(fruit[1:5], pattern = "apricot", replacement = "adbominable fruit")
```

```
## [1] "apple"              "adbominable fruit" "avocado"
## [4] "banana"             "bell pepper"
```

```r
# Replace all p's in the first seven elements with b's
str_replace_all(fruit[1:7], pattern = "p", replacement = "b")
```

```
## [1] "abble"       "abricot"     "avocado"     "banana"      "bell bebber"
## [6] "bilberry"    "blackberry"
```

Please format this Thanks

The function str_pad() can be used to make shorter strings into longer ones with spaces. This looks if you are trying to create a table with numbers. You give the string, the minimum width, a side (left, right or both) and a pad (default to whitespace). The function str_trim() can then remove these additional spaces. But there is no pad input for trim! :(

```r
max_width <- max(nchar(fruit[1:30])) # Longest width of first thirty elements

# add spaces to all fruit on the left side to match max_width
str_pad(fruit[1:30], max_width, "left")
```

```
##  [1] "       apple" "     apricot" "     avocado" "      banana"
##  [5] " bell pepper" "    bilberry" "  blackberry" "blackcurrant"
##  [9] "blood orange" "   blueberry" " boysenberry" "  breadfruit"
## [13] "canary melon" "  cantaloupe" "   cherimoya" "      cherry"
## [17] "chili pepper" "  clementine" "  cloudberry" "     coconut"
## [21] "   cranberry" "    cucumber" "     currant" "      damson"
## [25] "        date" " dragonfruit" "      durian" "    eggplant"
## [29] "  elderberry" "      feijoa"
```

```r
# add spaces to all fruit on the right side to match max_width
str_pad(fruit[1:30], max_width, "right")
```

```
##  [1] "apple       " "apricot     " "avocado     " "banana      "
##  [5] "bell pepper " "bilberry    " "blackberry  " "blackcurrant"
##  [9] "blood orange" "blueberry   " "boysenberry " "breadfruit  "
## [13] "canary melon" "cantaloupe  " "cherimoya   " "cherry      "
## [17] "chili pepper" "clementine  " "cloudberry  " "coconut     "
## [21] "cranberry   " "cucumber    " "currant     " "damson      "
## [25] "date        " "dragonfruit " "durian      " "eggplant    "
## [29] "elderberry  " "feijoa      "
```

```r
# add spaces to all fruit on the both sides to match max_width
str_pad(fruit[1:30], max_width, "both")
```

```
##  [1] "   apple    " "  apricot   " "  avocado   " "   banana   "
##  [5] "bell pepper " "  bilberry  " " blackberry " "blackcurrant"
##  [9] "blood orange" " blueberry  " "boysenberry " " breadfruit "
## [13] "canary melon" " cantaloupe " " cherimoya  " "   cherry   "
## [17] "chili pepper" " clementine " " cloudberry " "  coconut   "
## [21] " cranberry  " "  cucumber  " "  currant   " "   damson   "
## [25] "    date    " "dragonfruit " "   durian   " "  eggplant  "
## [29] " elderberry " "   feijoa   "
```

```r
# add dashes to all fruit on the left side to match max_width
str_pad(fruit[1:30], max_width, "left", pad = "-")
```

```
##  [1] "-------apple" "-----apricot" "-----avocado" "------banana"
##  [5] "-bell pepper" "----bilberry" "--blackberry" "blackcurrant"
##  [9] "blood orange" "---blueberry" "-boysenberry" "--breadfruit"
## [13] "canary melon" "--cantaloupe" "---cherimoya" "------cherry"
## [17] "chili pepper" "--clementine" "--cloudberry" "-----coconut"
## [21] "---cranberry" "----cucumber" "-----currant" "------damson"
## [25] "--------date" "-dragonfruit" "------durian" "----eggplant"
## [29] "--elderberry" "------feijoa"
```

```r
# remove all leading and trailing spaces from specific side
str_trim(str_pad(fruit[1:30], max_width, "left"))
```

```
##  [1] "apple"        "apricot"      "avocado"      "banana"
##  [5] "bell pepper"  "bilberry"     "blackberry"   "blackcurrant"
##  [9] "blood orange" "blueberry"    "boysenberry"  "breadfruit"
## [13] "canary melon" "cantaloupe"   "cherimoya"    "cherry"
## [17] "chili pepper" "clementine"   "cloudberry"   "coconut"
## [21] "cranberry"    "cucumber"     "currant"      "damson"
## [25] "date"         "dragonfruit"  "durian"       "eggplant"
## [29] "elderberry"   "feijoa"
```

If you are trying to make a long charactersting look like a paragraph, you are in luck! The function str_wrap() will take a string and format it into a paragraph with a specified width, indent and exdent.

```
paragraph <- "This paragraph is meant to be absolutely meaningless. If you don't approve of it, I'm going to be ve
ry upset with you. I spent a lot of time writing the rest of this post, so you better like my sense of humor"

# Make my long string into a paragraph with width 50
rf_para <- str_wrap(paragraph, width = 50)
cat(rf_para, "\n")
```

```
## This paragraph is meant to be absolutely
## meaningless. If you don't approve of it, I'm going
## to be very upset with you. I spent a lot of time
## writing the rest of this post, so you better like
## my sense of humor
```

```
# Make my long string into a paragraph with width 50, and an indent of 10
rf_para <- str_wrap(paragraph, width = 50, indent = 10)
cat(rf_para, "\n")
```

```
##           This paragraph is meant to be absolutely
## meaningless. If you don't approve of it, I'm going
## to be very upset with you. I spent a lot of time
## writing the rest of this post, so you better like
## my sense of humor
```

```
# Make my long string into a paragraph with width 50, and an exdent of 10
rf_para <- str_wrap(paragraph, width = 50, exdent = 10)
cat(rf_para, "\n")
```

```
## This paragraph is meant to be absolutely
##           meaningless. If you don't approve of it, I'm going
##           to be very upset with you. I spent a lot of time
##           writing the rest of this post, so you better like
##           my sense of humor
```

The last function we will review is str_view(). This function can be used to see highlighted patterns within a string! The function str_view_all() will show all patterns within a string. Use the match argument to choose if you only want strings with the specified pattern!

```
str_view(paragraph, "a") # Find first occurence of letter a in the string
```

```
This paragraph is meant to be absolutely meaningless. If you don't approve of it, I'm going to be very upset with
you. I spent a lot of time writing the rest of this post, so you better like my sense of humor
```

```
str_view_all(paragraph, "a") # Find all occurences of letter a in the string
```

```
This paragraph is meant to be absolutely meaningless. If you don't approve of it, I'm going to be very upset
with you. I spent a lot of time writing the rest of this post, so you better like my sense of humor
```

```
# Find all occurence of letter a in each string, exclude elements without a
str_view_all(fruit[1:10], "a", match = TRUE)
```

```
apple
apricot
avocado
banana
blackberry
blackcurrant
blood orange
```

Regular Expressions

> A regular expression, regex is, in theoretical computer scienceand formal language theory, a sequence of characters that define a search pattern.

- Found from Regular_expression article on wiki

In other words, regular expressions are essentially patterns that use a specific syntax for finding strings of interest. Check it out.

Metacharacters

Metacharacters are characters in regular expressions that take on special meaning. [] - Denote a set to becompared under. Good if you want to make a variety of letters to compare with. So if you want to to find all text with vowels, you can place all vowels in the set and the output will select any of the vowels. You can also place letters before or after, and you'll see that all patterns will be matched that will completed in that regular expression. This is why we don't need vectors for patterns, we can create the same affect of selecting multiple patterns by using the bracketr.

```
str_view_all(paragraph, "[aeiou]") # Find all vowels in the paragraph
```

```
This paragraph is meant to be absolutely meaningless. If you don't approve of it, I'm going to be very upset with you. I spent a lot of time writing the rest of this post, so you better like my sense of humor
```

```
str_view_all(paragraph, "m[ea]n") # Find all men and man substrings in text
```

```
This paragraph is meant to be absolutely meaningless. If you don't approve of it, I'm going to be very upset with you. I spent a lot of time writing the rest of this post, so you better like my sense of humor
```

```
# Find all rap, rep, and rip in the paragraph.
str_view_all(paragraph, "r[aei]p")
```

```
This paragraph is meant to be absolutely meaningless. If you don't approve of it, I'm going to be very upset with you. I spent a lot of time writing the rest of this post, so you better like my sense of humor
```

. - Any character. Loses it's behavior that it loses behavior in the set.

```
str_view_all(paragraph, ".") # Find all characters
```

```
This paragraph is meant to be absolutely meaningless. If you don't approve of it, I'm going to be very upset with you. I spent a lot of time writing the rest of this post, so you better like my sense of humor
```

```
str_view_all(paragraph, ".a") # Find all character then a subsgtrings
```

```
This paragraph is meant to be absolutely meaningless. If you don't approve of it, I'm going to be very upset with you. I spent a lot of time writing the rest of this post, so you better like my sense of humor
```

```
str_view_all(paragraph, "[.]") # Find all . characters in the paragraph
```

```
This paragraph is meant to be absolutely meaningless. If you don't approve of it, I'm going to be very upset with you. I spent a lot of time writing the rest of this post, so you better like my sense of humor
```

^ - Several cases 1. At the start of the pattern outside brackets - pattern starting with the following letters 2. At the start of a set - the negation of the rest of the set 3. At the end of a set - the negation of all but what is in the set

```
str_view_all(fruit[1:10], "^a") # Find all elements that start with a
```

```
apple
apricot
avocado
banana
bell pepper
bilberry
blackberry
blackcurrant
blood orange
blueberry
```

```
str_view_all(paragraph, "[^a]") # Find all characters that aren't a
```

```
This paragraph is meant to be absolutely meaningless. If you don't approve of it, I'm going to be very upset with you. I spent a lot of time writing the rest of this post, so you better like my sense of humor
```

```
str_view_all(paragraph, "[^aeiou]") # Find all consonants
```

```
This paragraph is meant to be absolutely meaningless. If you don't approve of it, I'm going to be very upset with you. I spent a lot of time writing the rest of this post, so you better like my sense of humor
```

```
str_view_all(paragraph, "[aeiou^]") # Find only vowels (just like [aeiou])
```

```
This paragraph is meant to be absolutely meaningless. If you don't approve of it, I'm going to be very upset with you. I spent a lot of time writing the rest of this post, so you better like my sense of humor
```

$ - end of a string

```
str_view_all(paragraph, "r$") # Does paragraph end with r?
```

This paragraph is meant to be absolutely meaningless. If you don't approve of it, I'm going to be very upset with you. I spent a lot of time writing the rest of this post, so you better like my sense of humor

```
str_view_all(fruit[1:25], "r$", match = TRUE) # All elements that end with r
```

bell pepper

chili pepper

cucumber

() - Used for grouping. Can create different patterns based on the structure of what is within the parenthesis.

- Is essentially an or statement. Often used with grouping.

```
# Find all elements with substrings ape and re
str_view_all(fruit, "(ap|r)e", match = TRUE)
```

breadfruit

grape

grapefruit

redcurrant

```
# Find all elements that have ape, re, and ge substrings of first 25 elem
str_view_all(fruit[1:25], "(ap|r|g)e", match = TRUE)
```

blood orange

breadfruit

```
# Find all elements that have ape, re, and ge substrings (same as above)
# of first 25 elem
str_view_all(fruit[1:25], "ape|re|ge", match = TRUE)
```

blood orange

breadfruit

```
# Find all elements that have ap, r, and ge substrings if first 25 elem
str_view_all(fruit[1:25], "ap|r|ge", match = TRUE)
```

apple

apricot

bell pepper

bilberry

blackberry

blackcurrant

blood orange

blueberry

boysenberry

breadfruit

canary melon

cherimoya

cherry

chili pepper

cloudberry

cranberry

cucumber

currant

- {n,m} denotes the repetition of a character (set of characters, or group) from n to m. {n,} can be used to simply have a minimum, and {0,m} can be used to simply have a maximum.

```
# Find all occurences in paragraph that have 2 or more p's
str_view_all(paragraph, "p{2,}")
```

This paragraph is meant to be absolutely meaningless. If you don't approve of it, I'm going to be very upset with you. I spent a lot of time writing the rest of this post, so you better like my sense of humor

```
# Find all occurences in paragraph that have an a followed by 2 or more p's
str_view_all(paragraph, "ap{2,}")
```

This paragraph is meant to be absolutely meaningless. If you don't approve of it, I'm going to be very upset with you. I spent a lot of time writing the rest of this post, so you better like my sense of humor

```r
# Find all occurences in paragraph that have an a and p repeated twice
str_view_all(paragraph, "(ap){2,}")
```

This paragraph is meant to be absolutely meaningless. If you don't approve of it, I'm going to be very upset with you. I spent a lot of time writing the rest of this post, so you better like my sense of humor

```r
# Find all occurences in the string below with two or three consecutive h's
str_view_all("WWWWhhhatt???", "h{2,3}")
```

WWWWhhhatt???

```r
# Find all occurences in the string below with one or two consecutive h's
str_view_all("WWWWhhhatt???", "h{1,2}")
```

WWWWhhhatt???

```r
# Find all fruit that have two consecutive o's in their name
str_view_all(fruit, "o{2,2}", match = TRUE)
```

blood orange

gooseberry

- ◦ To denote a range of characters in a set []. Ranges are created as follows - upper case, lower case, and then digits!

```r
# Find all occurences of a in paragraph
str_view_all(paragraph, "a")
```

This paragraph is meant to be absolutely meaningless. If you don't approve of it, I'm going to be very upset with you. I spent a lot of time writing the rest of this post, so you better like my sense of humor

```r
# Find all occurences of upper case letters in paragraph
str_view_all(paragraph, "[A-Z]")
```

This paragraph is meant to be absolutely meaningless. If you don't approve of it, I'm going to be very upset with you. I spent a lot of time writing the rest of this post, so you better like my sense of humor

```r
# Find all occurences of lower case letters in paragraph
str_view_all(paragraph, "[a-z]")
```

This paragraph is meant to be absolutely meaningless. If you don't approve of it, I'm going to be very upset with you. I spent a lot of time writing the rest of this post, so you better like my sense of humor

```r
# Find all occurences of lower and upper case letters in paragraph
str_view_all(paragraph, "[A-Za-e]")
```

This paragraph is meant to be absolutely meaningless. If you don't approve of it, I'm going to be very upset with you. I spent a lot of time writing the rest of this post, so you better like my sense of humor

```r
# Create a numbered fruit vector with three vectors
seven_fruit <- paste(c(seq(1,7,1)), fruit[1:7], sep = " ")

# Highlight the top three fruit
str_view_all(seven_fruit, "[1-3]")
```

1 apple

2 apricot

3 avocado

4 banana

5 bell pepper

6 bilberry

7 blackberry

```r
# HIghlight the top 4 fruit that start with an a
str_view_all(seven_fruit, "[0-4] a")
```

1 apple

2 apricot

3 avocado

4 banana

5 bell pepper

6 bilberry

7 blackberry

\ - double backslash (escaping the metacharacter to use it as a normal character) OR accessing anchors (next).

```r
str_view_all(paragraph, "\\.") # Find all occurences of .
```

```
This paragraph is meant to be absolutely meaningless. If you don't approve of it, I'm going to be very upset with
you. I spent a lot of time writing the rest of this post, so you better like my sense of humor
```

```r
str_view_all(paragraph, "\\^") # Find all occurences of ^
```

```
This paragraph is meant to be absolutely meaningless. If you don't approve of it, I'm going to be very upset with
you. I spent a lot of time writing the rest of this post, so you better like my sense of humor
```

```r
str_view_all(paragraph, "\\|") # Find all occurences of |
```

```
This paragraph is meant to be absolutely meaningless. If you don't approve of it, I'm going to be very upset with
you. I spent a lot of time writing the rest of this post, so you better like my sense of humor
```

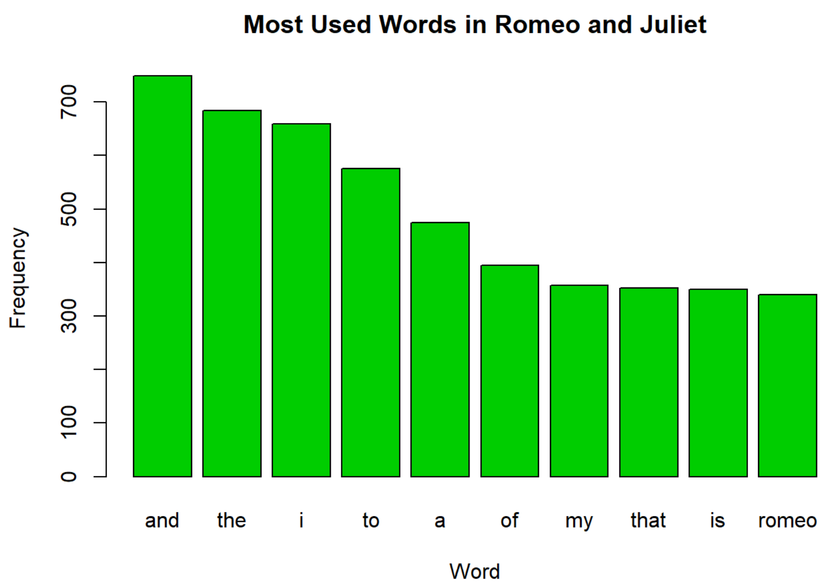
Examples

We are going to be working with the Romeo and Juliet play.

```r
# SHowing again fo rexample
Romeo_Juliet <- readLines(
 "http://www.textfiles.com/etext/AUTHORS/SHAKESPEARE/shakespeare-romeo-48.txt")
```

Finding the top 10 most frequently used words in Romeo and Juliet

```r
# Get rid of all non letter characters, by using negation for splitting
RJ_sentences <- strsplit(Romeo_Juliet, split = "[^A-Za-z]", perl = TRUE)

# unlist it for easier format
RJ_words <- unlist(RJ_sentences)

# Remove all empty vectors
RJ_words <- grep(pattern = "[A-Za-z]", RJ_words, perl = TRUE, value = TRUE)

# Change all words to lower case so they can be properly counted
RJ_words <- tolower(RJ_words)

#Plot it
barplot(sort(table(RJ_words), decreasing = TRUE)[1:10], xlab = "Word",
        ylab = "Frequency", main = "Most Used Words in Romeo and Juliet",
        col = 3)
```



Finding all words with length longer than 12 characters

```r
# Get rid of all non letter characters, by using negation for splitting
RJ_sentences <- strsplit(Romeo_Juliet, split = "[^A-Za-z]", perl = TRUE)

# unlist it for easier format
RJ_words <- unlist(RJ_sentences)

# Remove all empty vectors
RJ_words <- grep(pattern = "[A-Za-z]", RJ_words, perl = TRUE, value = TRUE)

# for nice viewing
RJ_words <- tolower(RJ_words)

# Subset of RJ_words using chars for words with longer than 12 characters
RJ_words <- RJ_words[nchar(RJ_words) > 12]

# Use union so that we only have unique words
RJ_words <- union(RJ_words, RJ_words)

# Display it
RJ_words
```

```
##  [1] "misadventured"  "interchanging"  "transgression"  "disparagement"
##  [5] "distemperature" "gentlemanlike"  "deliciousness"  "dishonourable"
##  [9] "unthankfulness" "uncomfortable"  "unsubstantial"
```

Change Romeo's name to Jack and Juliet's name to Jill

```r
# Use subsitution two times, both with regular expressions such that
# all cases of Romeo and Juliet are selected. Use extra spaces in replacement
# To try to maintain a good fomat
Jack_And_Jill <- sub("[Jj][Uu][Ll][Ii][Ee][Tt]", "JILL  ",
    sub("[Rr][Oo][Mm][Ee][Oo]", "JACK ", Romeo_Juliet, perl = TRUE),
    perl = TRUE)

# Print the first 100 lines of the new version of Jack and Jill
cat(Jack_And_Jill[1:100], sep = '\n')
```

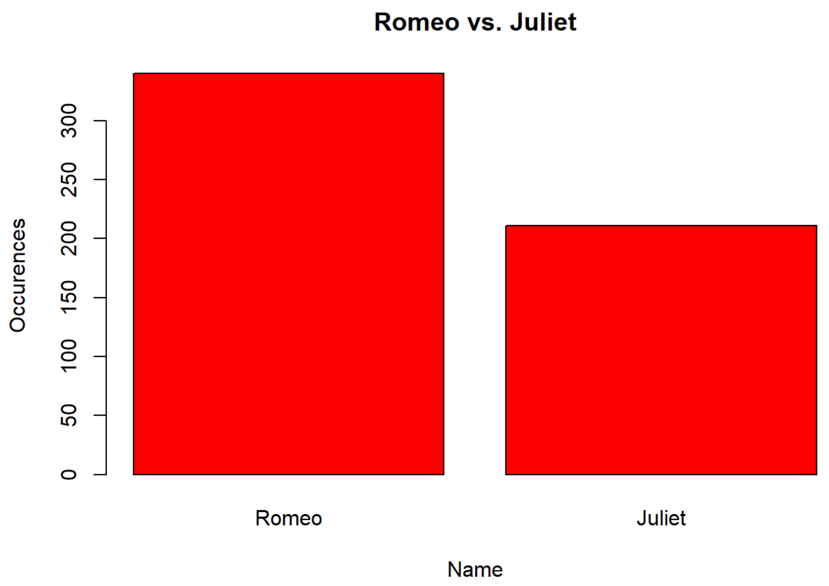
```
##  JACK  AND JILL
##
##  DRAMATIS PERSONAE
##
## ESCALUS  prince of Verona. (PRINCE:)
##
## PARIS    a young nobleman, kinsman to the prince.
##
## MONTAGUE |
##  |  heads of two houses at variance with each other.
## CAPULET  |
##
##  An old man, cousin to Capulet. (Second Capulet:)
##
## JACK     son to Montague.
##
## MERCUTIO kinsman to the prince, and friend to JACK .
##
## BENVOLIO nephew to Montague, and friend to JACK .
##
## TYBALT   nephew to Lady Capulet.
##
## FRIAR LAURENCE   |
##  |  Franciscans.
## FRIAR JOHN   |
##
## BALTHASAR    servant to JACK .
##
## SAMPSON  |
##  |  servants to Capulet.
## GREGORY  |
##
## PETER    servant to JILL  's nurse.
##
## ABRAHAM  servant to Montague.
##
##  An Apothecary. (Apothecary:)
##
##  Three Musicians.
##  (First Musician:)
##  (Second Musician:)
##  (Third Musician:)
##
##  Page to Paris; (PAGE:)  another Page; an officer.
##
## LADY MONTAGUE    wife to Montague.
```

```
##
## LADY CAPULET wife to Capulet.
##
## JILL     daughter to Capulet.
##
##  Nurse to JILL  . (Nurse:)
##
##  Citizens of Verona; several Men and Women,
##  relations to both houses; Maskers,
##  Guards, Watchmen, and Attendants.
##  (First Citizen:)
##  (Servant:)
##  (First Servant:)
##  (Second Servant:)
##  (First Watchman:)
##  (Second Watchman:)
##  (Third Watchman:)
##  Chorus.
##
## SCENE    Verona: Mantua.
##
##  JACK  AND JILL
##
##  PROLOGUE
##
##  Two households, both alike in dignity,
##  In fair Verona, where we lay our scene,
##  From ancient grudge break to new mutiny,
##  Where civil blood makes civil hands unclean.
##  From forth the fatal loins of these two foes
##  A pair of star-cross'd lovers take their life;
##  Whole misadventured piteous overthrows
##  Do with their death bury their parents' strife.
##  The fearful passage of their death-mark'd love,
##  And the continuance of their parents' rage,
##  Which, but their children's end, nought could remove,
##  Is now the two hours' traffic of our stage;
##  The which if you with patient ears attend,
##  What here shall miss, our toil shall strive to mend.
##
##  JACK  AND JILL
##
## ACT I
##
## SCENE I  Verona. A public place.
##
##  [Enter SAMPSON and GREGORY, of the house of Capulet,
##  armed with swords and bucklers]
##
## SAMPSON  Gregory, o' my word, we'll not carry coals.
##
## GREGORY  No, for then we should be colliers.
##
## SAMPSON  I mean, an we be in choler, we'll draw.
```

```
# Let's make a new file for it
cat(Jack_And_Jill, sep = "\n", file = "Jack_and_Jill.txt")
```

How many times are Romeo and Juliet's names used?

```
# Get rid of all non letter characters, by using negation for splitting
RJ_sentences <- strsplit(Romeo_Juliet, split = "[^A-Za-z]", perl = TRUE)

# unlist it for easier format
RJ_words <- unlist(RJ_sentences)

# Remove all empty vectors
RJ_words <- grep(pattern = "[A-Za-z]", RJ_words, perl = TRUE, value = TRUE)

# for nice viewing
RJ_words <- tolower(RJ_words)

# Use grep to find all occurrences of romeo, Juliet too
Romeo <- grep("romeo", RJ_words, value = TRUE)
Juliet <- grep("juliet", RJ_words, value = TRUE)

# Creating a vector for the barplot
RJ_Count <- c("Romeo" = length(Romeo), "Juliet" = length(Juliet))

# Bar Plot
barplot(RJ_Count, xlab = "Name", ylab = "Occurences", main = "Romeo vs. Juliet",
        col = 2)
```



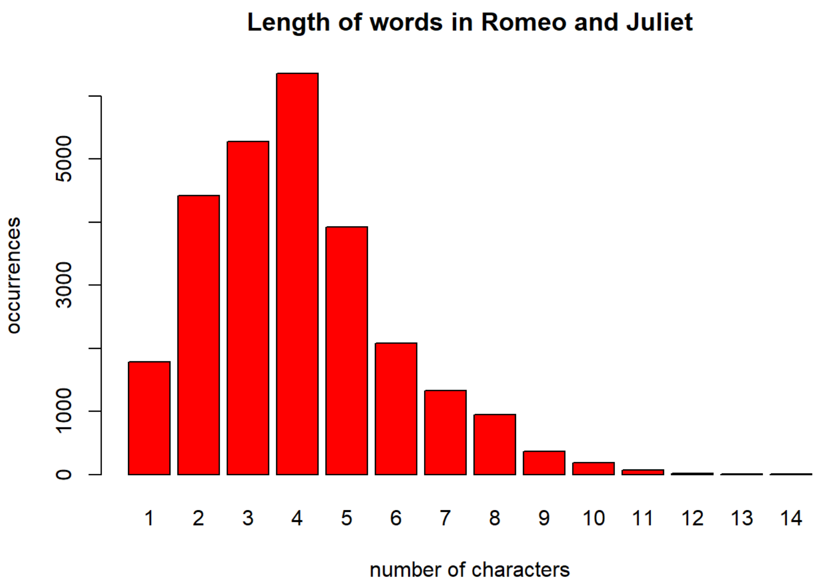
Tell me what fruit they ate in Romeo and Juliet

```
# Get rid of all non letter characters, by using negation for splitting
RJ_sentences <- strsplit(Romeo_Juliet, split = "[^A-Za-z]", perl = TRUE)

# unlist it for easier format
RJ_words <- unlist(RJ_sentences)

# Remove all empty vectors
RJ_words <- grep(pattern = "[A-Za-z]", RJ_words, perl = TRUE, value = TRUE)

# for nice viewing
RJ_words <- tolower(RJ_words)

# Find the intersection of both
Fruit_RJ <- intersect(RJ_words, fruit)

# That's a nice list
Fruit_list <- paste("They ate ", Fruit_RJ, "s.", sep = "", collapse = " ")

# Export it to a special file
cat(Fruit_list, file = "fruits.txt")
Fruit_list
```

```
## [1] "They ate dates. They ate nuts. They ate pears. They ate pomegranates."
```

Comparing the lengths of different words

```
# Get rid of all non letter characters, by using negation for splitting
RJ_sentences <- strsplit(Romeo_Juliet, split = "[^A-Za-z]", perl = TRUE)

# unlist it for easier format
RJ_words <- unlist(RJ_sentences)

# Remove all empty vectors
RJ_words <- grep(pattern = "[A-Za-z]", RJ_words, perl = TRUE, value = TRUE)

# Converting it to a numeric character
RJ_word_length <- str_length(RJ_words)

# Make a factor
RJ_word_length <- table(factor(RJ_word_length))

# Let's see
barplot(RJ_word_length, xlab = "number of characters", ylab = "occurrences",
        main = "Length of words in Romeo and Juliet", col = 2)
```



A Unique list of all words used in Romeo and Juliet

```
# Get rid of all non letter characters, by using negation for splitting
RJ_sentences <- strsplit(Romeo_Juliet, split = "[^A-Za-z]", perl = TRUE)

# unlist it for easier format
RJ_words <- unlist(RJ_sentences)

# Remove all empty vectors
RJ_words <- grep(pattern = "[A-Za-z]", RJ_words, perl = TRUE, value = TRUE)

# simplify comparisons
RJ_words <- tolower(RJ_words)

# Eliminate non-unique words
RJ_words <- intersect(RJ_words, RJ_words)

# Export this to another file
cat(RJ_words, sep = "\n", file = "RJWords.txt")
```

Summary of Procedure to Simple Text Processing

1. Import text using readLines() OR take existing data and coerce it into strings.
2. Create smaller strings using strsplit or substring/str_sub.
3. Use the case functions to equalize the data (if necessary)
4. Use Paste to recombine strings (if necessary).
5. Use regular expressions and sub(), gsub(), to create subsets.
6. Compare the subsets using setdiff(), union(), intersect() and other set related functions.
7. Reformat the data with str_wrap, str_trim, and str_pad.
8. Use cat to export the data.

Analyzing text is not very difficult with R. Just use these functions and you should be in good shape. I hope this summary will help you.

References:

1. http://www.gastonsanchez.com/Handling_and_Processing_Strings_in_R.pdf
2. http://www.gastonsanchez.com/r4strings/
3. http://r4ds.had.co.nz/strings.html
4. http://www.mjdenny.com/Text_Processing_In_R.html
5. https://en.wikibooks.org/wiki/R_Programming/Text_Processing
6. https://en.wikipedia.org/wiki/Regular_expression
7. http://opensourceforu.com/2017/02/text-mining-with-r/
8. https://www.stat.auckland.ac.nz/~paul/ItDT/HTML/node84.html
9. http://www.textfiles.com/etext/AUTHORS/SHAKESPEARE/shakespeare-romeo-48.txt
10. http://thomasleeper.com/Rcourse/Tutorials/stringmanipulation.html