# Lab04 - Data Types and QC

*Instructor*

*September 13, 2018*



# Learning Objectives

1. Identify the basic data types.
2. Import and export text files with Excel
3. QC data with data validation in Excel
4. Read text files into R as a **tibble**
5. Identify and convert suitable variables to a **factor**

# Preparation

1. Copy the folder `Lab04_DataTypes` from the R drive of the lab computer to your Research course home directory **AND** to the desktop of the lab computer.
2. Open the html version and Rmd version of Lab04.

# Data Types

When you import data from text files into Excel, there are only three possible data types: General, Text and Date. This is a real flaw of Excel. Users often rely on the General data type to import their data. This can lead to the corruption of the data. Two notorious examples are the inadvertant conversion of text values to dates and the loss of leading zero to numerical ID numbers. In addition, the General import allows each cell in a column to have a different data type!

With R, there are more data types, and the import process can be very controlled. The most common R data types are **character** (text), **numeric**, **integer**, **logical**, **factor**, **NA** (Not Available), and **date-time**. Importantly, these data types refer to how the data is stored in R and do not necessarily align with the data types discussed in lecture such as experiment or observational.

There are a large number of functions that allow you to read text files into R. In this class, we are going to use packages and functions from tidyverse (http://r4ds.had.co.nz/). The advantages of using **tidyverse** is that the functions have a more uniform syntax, many common tasks have been simplified, and the packages are well-integrated with RStudio.

# Excel for Data Collection

Excel is still a useful tool for certain tasks. Used with proper planning, it is useful for data collection or sharing. To plan properly, it help to understand what Excel can do to your data.

# Demonstration of Inadvertant Conversion of Data in Excel

1. Open `Example-01-General.xlsx` and examine the data.

| | A | B | C | D | E |
|---|---|---|---|---|---|
| 1 | id1 | date | text | id | number |
| 2 | 10.10.2010 | 10/10/2010 | 10-Oct | 10 | 10 |
| 3 | 10/10/2010 | 11/10/2010 | 10-Nov | 100 | 100 |
| 4 | 10_10_2010 | 12/10/2010 | 10-Dec | 1000 | 1000 |
| 5 | 10102010 | 2010/13/10 | roc10 | 1001 | 1001 |
| 6 | 10112010 | 1/10/2011 | 11-Jan | 1010 | |

2. Open `Example-01-Supervised.xlsx` and compare with the first file.

| | A | B | C | D | E |
|---|---|---|---|---|---|
| 1 | id1 | date | text | id | number |
| 2 | 10.10.2010 | 10/10/2010 | oct10 | 0010 | 10 |
| 3 | 10-10-2010 | 11/10/2010 | nov10 | 0100 | 100 |
| 4 | 10_10_2010 | 12/10/2010 | dec10 | 1000 | 1000 |
| 5 | 10102010 | 2010/13/10 | roc10 | 1001 | 1001 |
| 6 | 10112010 | 1/10/2011 | jan11 | 1010 | |

What happened? Column A should contain variations of a numeric identifier, but Excel but with the General import method, Excel has converted some to dates and some to numbers. In the second file, all are stored as text. Column B stores dates which have been imported faithfully, although one value has a data entry error that caused problems. Column C should contain nominal variables, but the General import method has converte several to dates. Column D contains numeric identifiers with important leading zeroes. The General import method has eliminated these. Column E contains integer values that were imported correctly both times.

3. Follow along as I import `Example-01-Raw.txt` with the Excel text import wizard.

*Instructor does a live demonstration for Excel Text Import Wizard. Important point is to avoid General import as much as possible to minimize data corruption.*

4. Import `Example-01-Raw.txt` in R. Do not worry about understanding the computer code now. You will have time later.

```
suppressMessages(library(tidyverse))
```

```
## Warning: package 'tidyverse' was built under R version 3.4.4
```

```
## Warning: package 'tidyr' was built under R version 3.4.4
```

```
## Warning: package 'dplyr' was built under R version 3.4.4
```

```
## Warning: package 'stringr' was built under R version 3.4.4
```

```
example01 <- read_delim("Example-01-Raw.txt", delim="\t")
```

```
## Parsed with column specification:
## cols(
##   id1 = col_character(),
##   date = col_date(format = ""),
##   text = col_character(),
##   id = col_character(),
##   number = col_integer()
## )
```

```
## Warning in rbind(names(probs), probs_f): number of columns of result is not
## a multiple of vector length (arg 2)
```

```
## Warning: 1 parsing failure.
## row # A tibble: 1 x 5 col     row col   expected   actual     file                 expected   <int
> <chr> <chr>      <chr>      <chr>               actual 1     4 date  valid date 2010/13/10 'Exampl
e-01-Raw.txt' file # A tibble: 1 x 5
```

Note the information provided by R. It tells you how each column of data was imported. Unlike Excel, R requires that values in the same column must have the same data type. There is also a warning about an invalid entry.

We can display the data.

```
example01
```

```
## # A tibble: 5 x 5
##   id1        date       text  id    number
##   <chr>      <date>     <chr> <chr>  <int>
## 1 10.10.2010 2010-10-10 oct10 0010      10
## 2 10-10-2010 2010-11-10 nov10 0100     100
## 3 10_10_2010 2010-12-10 dec10 1000    1000
## 4 10102010   NA         roc10 1001    1001
## 5 10112010   2011-01-10 jan11 1010      NA
```

R did a better job importing this data without intervention. Importantly, note the `<NA>` value. This is an invaluable data type in R. In this case, one value in the data column has a data entry error. We could go back to the text file to try to correct the issue. Compare this to the `NA` value for the missing number. This simply indicates that we have missing data.

We can get a quick summary of the data that was imported.

```
summary(example01)
```

```
##      id1                 date               text
##  Length:5           Min.   :2010-10-10   Length:5
##  Class :character   1st Qu.:2010-11-02   Class :character
##  Mode  :character   Median :2010-11-25   Mode  :character
##                     Mean   :2010-11-25
##                     3rd Qu.:2010-12-17
##                     Max.   :2011-01-10
##                     NA's   :1
##       id                 number
##  Length:5           Min.   :  10.0
##  Class :character   1st Qu.:  77.5
##  Mode  :character   Median : 550.0
##                     Mean   : 527.8
##                     3rd Qu.:1000.2
##                     Max.   :1001.0
##                     NA's   :1
```

This is descriptive information about the values in each column. Again, `NA` values are noted.

# Exercise - Excel Data Import

1. Locate the files `exampleGradeBookCSV.csv` and `exampleGradeBookTAB.txt` .
2. Double-click on the `exampleGradeBookCSV.csv` file. Excel should start and open the file.
3. Look in the first column of the file labeled “student”. Confirm that this is comprised of three letters and a number.
4. Examine rows 33 and 43. What happened here? The original data was JUN2 and MAR1.
5. Excel is notorious for changing certain numbers and text to dates using its General import method.
6. We can try to fix this problem. Highlight cell A33. Go to the Cells section of the Menu. Select Format > Format Cells > Text. Did it fix the problem?
7. We are done with this file. Close Excel and do not save the file.
8. Locate the file exampleGradeBookTAB. Double-click on it. It should open with NotePad or WordPad. It should not open with Excel. However, Excel can open this type of file. Close NotePad/WordPad.
9. Locate Excel on your lab computer, and open it. In the bottom left of the window is should have an option to “Open Other Workbooks”. Click on this and Browse to the folder for the workshop. Only Excel files should appear in the Window.
10. At the bottom right of this window is a drop down that displays “All Excel Files”. Change this to “All Files”. You should now see all files in the folder.
11. Select exampleGradeBookTAB. The Text Import Wizard should open.
12. In Step 1, confirm that file type is set to “Delimited”, and check the “My data has headers” box. Hit “Next”. Note the lines with the hash signs. This contains metadata about this data set. It is always a good idea to include metadata at the beginning of a file, marking it with a specific character or flag.
13. In Step 2, confirm that the “Delimiter” is set to Tab and the “Text qualifier” is set to none. If you want, you can change these settings to see how it alters the “Data preview”. Make sure that you select the appropriate settings before hitting “Next”.
14. In Step 3, change the “Column data format” to Text for the columns labelled “student”, “class” and “letter.grade”.
15. Examine rows 33 and 43. The student identifiers should now be correct because we did not allow Excel to guess at the data in this column.

# Exercise - Data Validation

1. The “class” column appeared to have some typos. These would be pretty easy to find and fix manually with this small file, but there is a better way.
2. Make a new worksheet called validation. In cells A1 through A5, enter class, Freshman, Sophomore, Junior, Senior. We are going to use this small list for Data Validation.

3. Highlight the data in the class column by selecting cell B2. Hold down Control and Shift and hit the Down Arrow. All data in column B should now be highlighted.
4. In the Data tab, find the Data Validation icon. It may or may not be labelled depending on the size of your window. The icon appears as two small rectangles, one with a check and one with a red circle/slash. Click on the icon and select “Data Validation”.
5. In the “Allow” window, select List.
6. In the “Source” box, click on the small spreadsheet icon at the right. Select cells A2 through A5 in the validation worksheet and click on the spreadsheet icon again.
7. Click on “OK” in the Data Validation window. Nothing seems to change in the “class” column.
8. Make sure that all data in the “class” column is still highlighted. Go to Data Validation again and select “Circle invalid data”.
9. Scroll down, and you should see the invalid entries in the “class” column. You can fix these by selecting the bad cells and choosing the valid values from the drop down.

Data validation is an excellent way to check an existing data set, but you can also use it to control data entry as well. Examine the possiblities in the Data Validation window. What strategies could you use on the other columns? Imagine that you had six undergraduates helping you collect data for an important experiment. How could you use Excel Data Validation to control the possible chaos?

# Importing Data into R

Data collection or entry is not a strength of R. There are packages for this, but they tend to be cumbersome. Most analyses with R start with a text files containing data that is imported into R. In addition, there are packages that allow you to read in other formats such as Excel, SAS and SPSS.
Today, we are going to read in our data as a **tibble**, a new class of R object related to a **data frame**. You can think of these as the equivalent of an Excel worksheet but with important differences. In a **tibble** or **data frame**, rows are records and columns are variables. Unlike Excel, the values in a column **MUST** be the same data type. In addition, there must be a valid value for every variable. If no valid variable is found, R will introduce an NA.

Data validation in R is more flexible and more complex that data validation in Excel. Today, we will use

```
grades <- read_delim("exampleGradeBookTAB.txt", delim="\t", comment="#")
```

```
## Parsed with column specification:
## cols(
##   student = col_character(),
##   class = col_character(),
##   test1 = col_integer(),
##   test2 = col_integer(),
##   test3 = col_integer(),
##   attend = col_integer(),
##   attend.perc = col_integer(),
##   final.score = col_double(),
##   adj.final.score = col_double(),
##   letter.grade = col_character()
## )
```

The data appears to have been read in with no problems.

```
grades
```

```
## # A tibble: 100 x 10
##    student class     test1 test2 test3 attend attend.perc final.score
##    <chr>   <chr>     <int> <int> <int>  <int>       <int>       <dbl>
##  1 BAD30   Sophomore    48    65    81     40          89        73.9
##  2 BAG69   Junior       70    42    74     42          93        67.9
##  3 BAP77   Senior       76    58    71     42          93        71.1
##  4 BIC74   Junior       75    85    72     42          93        78.2
##  5 BIG44   Sophomore    55    76    79     40          89        76.5
##  6 BIW87   Senior       81    80    68     43          96        76.1
##  7 BIX12   Freshman     35    48    89     39          87        71.8
##  8 BOJ29   Sophomore    46    90    81     40          89        79.9
##  9 BON82   Senior       79    72    70     43          96        74.9
## 10 BOP78   Senior       77    43    71     42          93        67.5
## # ... with 90 more rows, and 2 more variables: adj.final.score <dbl>,
## #   letter.grade <chr>
```

A summary of the data could reveal problems.

```
summary(grades)
```

```
##    student             class               test1            test2
##  Length:100         Length:100         Min.   :  4.00   Min.   : 23.00
##  Class :character   Class :character   1st Qu.: 45.00   1st Qu.: 56.75
##  Mode  :character   Mode  :character   Median : 56.00   Median : 71.00
##                                        Mean   : 58.96   Mean   : 68.95
##                                        3rd Qu.: 75.25   3rd Qu.: 84.00
##                                        Max.   :103.00   Max.   :120.00
##      test3            attend       attend.perc      final.score
##  Min.   : 53.00   Min.   :33.00   Min.   : 73.00   Min.   :61.62
##  1st Qu.: 71.75   1st Qu.:39.00   1st Qu.: 87.00   1st Qu.:71.75
##  Median : 78.00   Median :41.00   Median : 91.00   Median :74.88
##  Mean   : 77.65   Mean   :40.58   Mean   : 90.25   Mean   :74.71
##  3rd Qu.: 82.25   3rd Qu.:42.00   3rd Qu.: 93.00   3rd Qu.:78.25
##  Max.   :105.00   Max.   :45.00   Max.   :100.00   Max.   :87.62
##  adj.final.score letter.grade
##  Min.   :69.62   Length:100
##  1st Qu.:79.75   Class :character
##  Median :82.88   Mode  :character
##  Mean   :82.71
##  3rd Qu.:86.25
##  Max.   :95.62
```

Three of our columns imported as characters and seven as numeric. There is some reason for concern about the test scores. Should students be able to get greater than 100? In this case, yes because this is simulated data, and I did not allow

One problem with the data as it stands is that we know that **class** and **grade** are categorical variables because there are only a limited number of valid values. In Excel, we made a 'list' of valid values for **class**. We can do something similar here. **Note that R is case sensitive!**

```
count(grades, class)
```

```
## # A tibble: 7 x 2
##   class         n
##   <chr>     <int>
## 1 Freshman     13
## 2 Jnior         1
## 3 Junior       29
## 4 Senior       22
## 5 Senor         2
## 6 Sophomor      1
## 7 Sophomore    32
```

We can definitely see that there are typos in class.

```
count(grades, letter.grade)
```

```
## # A tibble: 9 x 2
##   letter.grade     n
##   <chr>        <int>
## 1 A                4
## 2 A-               5
## 3 B               45
## 4 B-              17
## 5 B+               2
## 6 C               13
## 7 C-               3
## 8 C+              10
## 9 D+               1
```

This column appears to be OK.

```
class_values <- c("Freshman", "Sophomore", "Junior", "Senior")
class_values
```

```
## [1] "Freshman"  "Sophomore" "Junior"    "Senior"
```

We can now check the **class** column for valid entries. We are going to use a matching operator, `%in%` to do this. The result will be a logical vector, TRUE or FALSE. If the value of **class** in **grades** is a valid value, then `TRUE` will be returned, otherwise `FALSE` .

```
class_valid <- grades$class %in% class_values
class_valid
```

```
##   [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [12]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [23]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [34]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [45]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [56]  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [67]  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [78]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE
##  [89]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE
## [100]  TRUE
```

We can use this logical vector to examine the invalid entries. To do this, we need to look at entries that are FALSE. We need to use the **not** operator, `!` to do this.

```
!class_valid
```

```
##   [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [12] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [23] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [34] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [45] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [56] FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE
##  [67] FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE
##  [78] FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE
##  [89] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE
## [100] FALSE
```

Note how this simply reveresed the logical values. We can now display the rows in grades with invalid class values.

```
filter(grades, !class_valid)
```

```
## Warning: package 'bindrcpp' was built under R version 3.4.4
```

```
## # A tibble: 4 x 10
##   student class    test1 test2 test3 attend attend.perc final.score
##   <chr>   <chr>    <int> <int> <int>  <int>       <int>       <dbl>
## 1 RAS51   Jnior       56    89    78     41          91        79.6
## 2 SOQ91   Senor       84    29    66     43          96        62.8
## 3 XAH83   Senor       79    48    69     43          96        68.4
## 4 ZIK34   Sophomor    50    84    81     40          89        78.9
## # ... with 2 more variables: adj.final.score <dbl>, letter.grade <chr>
```

How do you fix these values? There is a way to edit this data graphically, but that manual manipulation of the data would not be documented. Better to actually code this. To do this, we can use `mutate` and `replace` to alter the values in the **tibble**.

```
grades <- mutate(grades, class=replace(class, student == "RAS51", "Junior"))
grades <- mutate(grades, class=replace(class, student == "SOQ91", "Senior"))
grades <- mutate(grades, class=replace(class, student == "XAH83", "Senior"))
grades <- mutate(grades, class=replace(class, student == "ZIK34", "Sophomore"))
filter(grades, !class_valid)
```

```
## # A tibble: 4 x 10
##   student class     test1 test2 test3 attend attend.perc final.score
##   <chr>   <chr>     <int> <int> <int>  <int>       <int>       <dbl>
## 1 RAS51   Junior       56    89    78     41          91        79.6
## 2 SOQ91   Senior       84    29    66     43          96        62.8
## 3 XAH83   Senior       79    48    69     43          96        68.4
## 4 ZIK34   Sophomore    50    84    81     40          89        78.9
## # ... with 2 more variables: adj.final.score <dbl>, letter.grade <chr>
```

This method would be inconvenient if you had a big data set with many problems, but so would manual editing. However, this way, we have clear documentation of what was done. In addition, with more programming experience, you can develop efficient R code to handle bigger problems.

We can convert both **class** and **letter.grade** to the R data type **factors**. Similar to Excel Data Validation, only specific values will be allowed for these variables. In addition, if the **factor** is an ordinal variable, we can specifiy the order.

```
grades <- mutate(grades, class=parse_factor(class, levels=class_values))
grades %>% count(class)
```

```
## # A tibble: 4 x 2
##   class         n
##   <fct>     <int>
## 1 Freshman     13
## 2 Sophomore    33
## 3 Junior       30
## 4 Senior       24
```

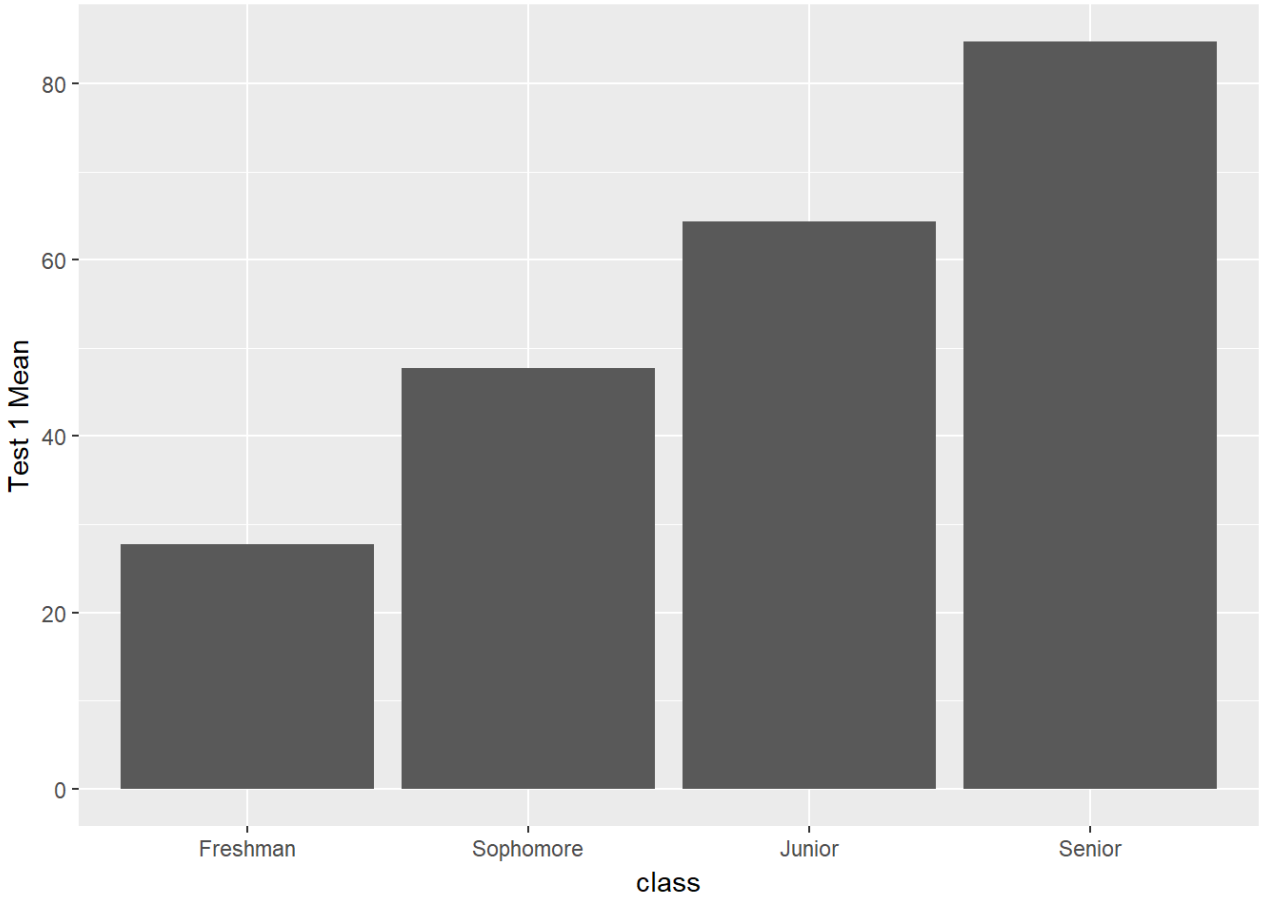
Note, that class is shown in the table by the order that we specified, not in alphabetical order (which is the default).

```
valid_grades <- c("A+", "A", "A-", "B+", "B", "B-", "C+", "C", "C-","D+", "D", "F")
grades <- mutate(grades, letter.grade=parse_factor(letter.grade, levels=valid_grades))
grades %>% count(letter.grade)
```

```
## # A tibble: 9 x 2
##   letter.grade     n
##   <fct>        <int>
## 1 A                4
## 2 A-               5
## 3 B+               2
## 4 B               45
## 5 B-              17
## 6 C+              10
## 7 C               13
## 8 C-               3
## 9 D+               1
```

The advantage of controlling the order for the factors in also apparent in the following plot.

```
grades %>%
  group_by(class) %>%
  summarise(`Test 1 Mean`=mean(test1, na.rm=TRUE)) %>%
  ggplot() +
  aes(x=class, y=`Test 1 Mean`) +
  geom_col()
```




We should save the edited version of our the grade book. Note, that the order for the **factors** can not be saved in the text file. However, we have saved the process in this file!

```
write_delim(grades, path="exampleGradeBookTAB_corrected.txt", delim="\t")
```

# Lab Exercise01

You will need the files “flights_ex.txt” and the datasets that we transferred during Lab 02.

Do the following:

1. Read in “flights_ex.txt” as a **tibble**
2. Determine which columns are suitable factors. Hint, use the **count()** function.
3. How can you get the valid values for these columns?
4. Check these columns for valid values.
5. Fix any bad entries. Assume that simple typos were made and make your best guess.
6. Convert these columns to factors.
7. Save the edited version to a file.

Write your code in an Rmd file with suitable comments. You should work in groups of 2 or 3, but each of you must turn in your own Rmd file. The code can be indentical, but the comments and narrative should be your own.

Due Date: Start of lab on Sept. 21. Place your Rmd file in your project, clearly labeled. This exercise is not a requirement of your Data Management Notebook. However, we do expect that some of the information from today will be incorporated into the notebook.

# Session Information

```
sessionInfo()
```

```
## R version 3.4.3 (2017-11-30)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 15063)
##
## Matrix products: default
##
## locale:
## [1] LC_COLLATE=English_United States.1252
## [2] LC_CTYPE=English_United States.1252
## [3] LC_MONETARY=English_United States.1252
## [4] LC_NUMERIC=C
## [5] LC_TIME=English_United States.1252
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] bindrcpp_0.2.2  forcats_0.3.0   stringr_1.3.1   dplyr_0.7.5
##  [5] purrr_0.2.4     readr_1.1.1     tidyr_0.8.1     tibble_1.4.2
##  [9] ggplot2_2.2.1   tidyverse_1.2.1
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_0.12.17     cellranger_1.1.0 pillar_1.2.3     compiler_3.4.3
##  [5] plyr_1.8.4       bindr_0.1.1      tools_3.4.3      digest_0.6.15
##  [9] lubridate_1.7.4  jsonlite_1.5     evaluate_0.10.1  nlme_3.1-131.1
## [13] gtable_0.2.0     lattice_0.20-35  pkgconfig_2.0.1  rlang_0.2.0
## [17] psych_1.8.4      cli_1.0.0        rstudioapi_0.7   yaml_2.1.19
## [21] parallel_3.4.3   haven_1.1.1      xml2_1.2.0       httr_1.3.1
## [25] knitr_1.20       hms_0.4.2        rprojroot_1.3-2  grid_3.4.3
## [29] tidyselect_0.2.4 glue_1.2.0       R6_2.2.2         readxl_1.1.0
## [33] foreign_0.8-70   rmarkdown_1.9    modelr_0.1.2     reshape2_1.4.3
## [37] magrittr_1.5     backports_1.1.2  scales_0.5.0     htmltools_0.3.6
## [41] rvest_0.3.2      assertthat_0.2.0 mnormt_1.5-5     colorspace_1.3-2
## [45] labeling_0.3     utf8_1.1.4       stringi_1.1.7    lazyeval_0.2.1
## [49] munsell_0.4.3    broom_0.4.4      crayon_1.3.4
```