Assignment 3

edelsonc

August 31, 2016

This assignment will concern itself with the download, loading, cleaning and counting data from a dataset from UC Irvine working with diabetic patients

Downloading the data

The data was downloaded using the `wget` command in bash. However, since it arrived as a zip folder, it had to be unzipped before it could be used.

```
url_data="https://archive.ics.uci.edu/ml/machine-learning-databases/00296/dataset_diabetes.zip"
wget "$url_data"
unzip ./dataset_diabetes.zip
echo "Dataset downloaded from $url_data at $( date )" > UCI_dataprovidence.txt
```

Now we can load the data into R and look at its struture

```
f_path <-"~/Desktop/Data_Science/EDA/assignment_3/dataset_diabetes/diabetic_data.csv"

diabetic_data <- read.csv(f_path)

class(diabetic_data)
```

```
## [1] "data.frame"
```

This informs us that our csv has been loaded into the file as a dataframe. Additionally, if we bothered to inspect the directory that we unzipped, we'd have noticed that a second csv was also present, `IDs_mapping.csv`. We can load this into `R` as well, and look at its structure.

```
id_path <-"~/Desktop/Data_Science/EDA/assignment_3/dataset_diabetes/IDs_mapping.csv"

id_keys <- read.csv(id_path)

class(id_keys)
```

```
## [1] "data.frame"
```

Once again, we see that `R` has imported the csv into a dataframe.

Determining and Documenting Missing Values

In order to determine the missing values, it is easiest to simply look at the data. Using the command lines `head` and `cut` command, we'll inspect the first 10×8 section of the csv

```
 f_path="/Users/edelsonc/Desktop/Data_Science/EDA/assignment_3/dataset_diabetes/diabetic_data.csv"

head "$f_path" | cut -d',' -f1-8
```

```
## encounter_id,patient_nbr,race,gender,age,weight,admission_type_id,discharge_disposition_id
## 2278392,8222157,Caucasian,Female,[0-10),?,6,25
## 149190,55629189,Caucasian,Female,[10-20),?,1,1
## 64410,86047875,AfricanAmerican,Female,[20-30),?,1,1
## 500364,82442376,Caucasian,Male,[30-40),?,1,1
## 16680,42519267,Caucasian,Male,[40-50),?,1,1
## 35754,82637451,Caucasian,Male,[50-60),?,2,1
## 55842,84259809,Caucasian,Male,[60-70),?,3,1
## 63768,114882984,Caucasian,Male,[70-80),?,1,1
## 12522,48330783,Caucasian,Female,[80-90),?,2,1
```

Looking at this, we can notice that there seem to be a large amount of question marks. After some poking around, we can determine that these represent missing data, as we see that there are "Nones" which have a meaning, and no whitespace.

Now that we know what a missing value looks like, we have to find a way to count it. A simple way is to use the `sapply` function on each column, checking with `is.element` how many elements are `"?"`. However, this is slow, and will require us to manually do this to every column. Instead, we can write a function to help us do this

```
sum_missing <- function(data_set, search_item){
 sum(sapply(data_set, is.element, el=search_item))
}
```

What this function does is simply check each element in a row or column for a given value, creates a list of booleans, and then sums those. `sum_missing` can now be used with the `apply` function columnwise in order to count the number of missing values in each column

```
missing_counts <- apply(diabetic_data, 2, sum_missing, search_item="?")

str(missing_counts)
```

```
##  Named int [1:50] 0 0 2273 0 0 98569 0 0 0 0 ...
##  - attr(*, "names")= chr [1:50] "encounter_id" "patient_nbr" "race" "gender" ...
```

This allows us to now sum `missing_counts`, giving us a final value of 192849 missing values.

Counting Values

Admission to the Emergency Room

Now that we have an idea and way of counting missing fields, we can begin to look at counts of other events. First, we'll count the number of patients admitted to the hospital via the emergency room. But how do we know if they are? We simply check `IDs_mapping.csv` to see which code corresponds in `admission_type_id`.

```
sum(diabetic_data$admission_type_id == 1)
```

```
## [1] 53990
```

We can find what percentage of the total admitted patients that is by dividing it by the length of `diabetic_data$admission_type_id` minus any missing values recorded in `missing_count`

```
sum(diabetic_data$admission_type_id == 1)/(length(diabetic_data$admission_type_id) - missing_counts[7])
```

```
## admission_type_id
##         0.5305308
```

A question of interest might be to know what percentage of the patients admitted to the emergency room leave the hospital as "expired". This can be checked by combining logic

```
num_exp <- sum((diabetic_data$admission_type_id == 1 & diabetic_data$discharge_disposition_id == 7))
num_emerg <- sum(diabetic_data$admission_type_id == 1)
```

Taking the ratio of these we get that 0.7723653% of patients admitted to the emergency room leave as "expired".

Common Admission types

To discovered what the most common admision type and the most common discharge type is, it is helpful to take a subsection of our data that only contains those two field

```
admin_dis <- diabetic_data[c(7,8)]
str(admin_dis)
```

```
## 'data.frame':    101766 obs. of  2 variables:
##  $ admission_type_id       : int  6 1 1 1 1 2 3 1 2 3 ...
##  $ discharge_disposition_id: int  25 1 1 1 1 1 1 1 1 3 ...
```

Now all we have to do is count what the most common status for both of those are. But there's a problem. If you look at the output of `str`, you'll notice that the data type in each column is `int`. If we wish to count the data, well have to turn them into factors or character

```
admin_dis$admission_type_id <- factor(admin_dis$admission_type_id)
admin_dis$discharge_disposition_id <- factor(admin_dis$discharge_disposition_id)
```

Applying `summary` to our new dataframe will give us a count for each catagory

```
summary(admin_dis)
```

```
##  admission_type_id discharge_disposition_id
##  1      :53990     1      :60234
##  3      :18869     3      :13954
##  2      :18480     6      :12902
##  6      : 5291     18     : 3691
##  5      : 4785     2      : 2128
##  8      :  320     22     : 1993
##  (Other):   31     (Other): 6864
```

So the most common admission is emergency, while the most frequent discharge status is to home.

To look at the the most common discharge id for the most common admission id, we can first create a vector just containing the discharge ids for the most common admission id, emergency room. Then we can take its summary and sort it in descending order.

```
 er_admin <- admin_dis[admin_dis$admission_type_id == "1",2]
sort(summary(er_admin), decreasing=TRUE)
```

```
##     1     3     6    18     2    11    22     5     4     7    13    23
## 31695  7813  6572  2142  1189  1102   960   587   512   417   278   257
##    14    28     8    15    24    25     9    27    19    17    12    20
##   235    79    43    38    37    12    10     5     3     2     1     1
##    10    16
##     0     0
```

Showing us that the most common discharge for an emergency room admission was returning to the family.

Distribution of Admission

Finally, we may want to look at the distribution of admission. This is easy to do in r, since it has many built in functions to produce simple graphics.

```
barplot(summary(admin_dis[[1]]), col = "skyblue", xlab="Admission ID", ylab="Count")
```