Assignment 8

Kornel Kovacs

10/31/2019

Exercise 1:

I created a project for this assignment in a folder.

Exercise 2:

All done

Exercise 3:

Reading in from the CSV:

```
df_csv <- read_csv('./data_repo/hotels-europe/clean/hotels-europe_price.csv')
```

```
## Parsed with column specification:
## cols(
##   hotel_id = col_double(),
##   price = col_double(),
##   offer = col_double(),
##   offer_cat = col_character(),
##   year = col_double(),
##   month = col_double(),
##   weekend = col_double(),
##   holiday = col_double(),
##   nnights = col_double(),
##   scarce_room = col_double()
## )
```

Reading in from the DTA:

```
df_dta <- read_dta('./data_repo/hotels-europe/clean/hotels-europe_price.dta')
```

Let us see whether they have the same lenght or not. Spoiler: They do.

```
nrow(df_csv)
```

```
## [1] 148021
```

```
nrow(df_dta)
```

```
## [1] 148021
```

Let us see whether they have the same head and tail or not. Spoiler: They do.

```
head(df_csv)
```

```
## # A tibble: 6 x 10
##   hotel_id price offer offer_cat  year month weekend holiday nnights
##      <dbl> <dbl> <dbl> <chr>     <dbl> <dbl>   <dbl>   <dbl>   <dbl>
## 1        1   172     0 0% no of~  2017    11       1       0       1
## 2        1   122     1 15-50% o~  2018     1       1       0       1
## 3        1   122     1 15-50% o~  2017    12       0       1       1
## 4        1   552     1 1-15% of~  2017    12       0       1       4
## 5        1   122     1 15-50% o~  2018     2       1       0       1
## 6        1   114     1 15-50% o~  2017    11       0       0       1
## # ... with 1 more variable: scarce_room <dbl>
```

```
head(df_dta)
```

```
## # A tibble: 6 x 10
##   hotel_id price scarce_room offer offer_cat  year month weekend holiday
##      <dbl> <dbl>       <dbl> <dbl> <chr>     <dbl> <dbl>   <dbl>   <dbl>
## 1        1   114           0     1 15-50% o~  2017    11       0       0
## 2        1   172           0     0 0% no of~  2017    11       1       0
## 3        1   122           0     1 15-50% o~  2017    12       0       1
## 4        1   552           0     1 1-15% of~  2017    12       0       1
## 5        1   122           0     1 15-50% o~  2018     1       1       0
## 6        1   122           0     1 15-50% o~  2018     2       1       0
## # ... with 1 more variable: nnights <dbl>
```

```
tail(df_csv)
```

```
## # A tibble: 6 x 10
##   hotel_id price offer offer_cat  year month weekend holiday nnights
##      <dbl> <dbl> <dbl> <chr>     <dbl> <dbl>   <dbl>   <dbl>   <dbl>
## 1    22902   109     1 15-50% o~  2018     2       1       0       1
## 2    22902   119     1 15-50% o~  2017    11       0       0       1
## 3    22902   109     1 15-50% o~  2018     4       1       0       1
## 4    22902   109     1 15-50% o~  2018     3       1       0       1
## 5    22902   446     1 15-50% o~  2017    12       0       1       4
## 6    22902   117     1 15-50% o~  2017    12       0       1       1
## # ... with 1 more variable: scarce_room <dbl>
```

```
tail(df_dta)
```

```
## # A tibble: 6 x 10
##   hotel_id price scarce_room offer offer_cat  year month weekend holiday
##      <dbl> <dbl>       <dbl> <dbl> <chr>     <dbl> <dbl>   <dbl>   <dbl>
## 1    22902   446           1     1 15-50% o~  2017    12       0       1
## 2    22902   117           1     1 15-50% o~  2017    12       0       1
## 3    22902   149           1     1 15-50% o~  2018     1       1       0
## 4    22902   109           1     1 15-50% o~  2018     2       1       0
## 5    22902   109           1     1 15-50% o~  2018     3       1       0
## 6    22902   109           1     1 15-50% o~  2018     4       1       0
## # ... with 1 more variable: nnights <dbl>
```

I was wondering whether we have the same columns and types of them are consistent or not. Spoiler: They do.

```
sapply(df_csv, class)
```

```
##    hotel_id       price       offer   offer_cat        year       month
##   "numeric"   "numeric"   "numeric" "character"   "numeric"   "numeric"
##     weekend     holiday     nnights scarce_room
##   "numeric"   "numeric"   "numeric"   "numeric"
```

```
sapply(df_dta, class)
```

```
##    hotel_id       price scarce_room       offer   offer_cat        year
##   "numeric"   "numeric"   "numeric"   "numeric" "character"   "numeric"
##       month     weekend     holiday     nnights
##   "numeric"   "numeric"   "numeric"   "numeric"
```

Finally, there are more sophisticated ways of showing whether they are identical or not. I should have started with this one, but I wanted to show several aspects of testing before coming up with these functions.

All values are the same or not:

```
all.equal(df_csv, df_dta)
```

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

Okay..but these are only the values. Are they dentical objects as well?

```
identical(df_dta, df_csv)
```

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

Somehow, they are not identical according to the function `identical()`

Exercise 4:

Creating a sample of 200 to write out.

```
df_csv_200 <- df_csv[sample(nrow(df_csv), 200), ]
```

Actually writing it out. Notice, I already did the alteration part as `write_csv2()` uses `;` as a separator.

```
write_csv2(df_csv_200, "hotels_cleaned_200.csv")
```

Now, try reading it back in.

```
df_csv_200 <- read_csv("hotels_cleaned_200.csv")
```

```
## Parsed with column specification:
## cols(
##   `hotel_id;price;offer;offer_cat;year;month;weekend;holiday;nnights;scarce_room` = col_character()
## )
```

```
head(df_csv_200)
```

```
## # A tibble: 6 x 1
##   `hotel_id;price;offer;offer_cat;year;month;weekend;holiday;nnights;scarc~
##   <chr>
## 1 11181; 30;0;0% no offer;2018; 1;1;0;1;0
## 2 14165;110;0;0% no offer;2017;12;0;1;1;0
## 3 11774;146;1;75%+ offer;2017;11;1;0;1;1
## 4 14410; 65;1;15-50% offer;2017;11;0;0;1;0
## 5 3768; 80;1;50%-75% offer;2017;11;0;0;1;1
## 6 8800; 71;1;50%-75% offer;2018; 4;1;0;1;0
```

We do not see a structure, just the whole thing in one column. The problem is that `read_csv()` is looking for `,` to separate values by, but it doesn't find any of such a kind. As a solution we either have to specify the value of the separator or use another function which has `;` as the default separator value. `read_csv2()` is such a function. I chose this way of screwing up the dataset because it is very often the case that we have something else as a separator other than `,`.

Exercise 5:

We have data from WHO to make it tidy.

```
head(who)
```

```
## # A tibble: 6 x 60
##   country iso2  iso3   year new_sp_m014 new_sp_m1524 new_sp_m2534
##   <chr>   <chr> <chr> <int>       <int>        <int>        <int>
## 1 Afghan~ AF    AFG    1980          NA           NA           NA
## 2 Afghan~ AF    AFG    1981          NA           NA           NA
## 3 Afghan~ AF    AFG    1982          NA           NA           NA
## 4 Afghan~ AF    AFG    1983          NA           NA           NA
## 5 Afghan~ AF    AFG    1984          NA           NA           NA
## 6 Afghan~ AF    AFG    1985          NA           NA           NA
## # ... with 53 more variables: new_sp_m3544 <int>, new_sp_m4554 <int>,
## #   new_sp_m5564 <int>, new_sp_m65 <int>, new_sp_f014 <int>,
## #   new_sp_f1524 <int>, new_sp_f2534 <int>, new_sp_f3544 <int>,
## #   new_sp_f4554 <int>, new_sp_f5564 <int>, new_sp_f65 <int>,
## #   new_sn_m014 <int>, new_sn_m1524 <int>, new_sn_m2534 <int>,
## #   new_sn_m3544 <int>, new_sn_m4554 <int>, new_sn_m5564 <int>,
## #   new_sn_m65 <int>, new_sn_f014 <int>, new_sn_f1524 <int>,
## #   new_sn_f2534 <int>, new_sn_f3544 <int>, new_sn_f4554 <int>,
## #   new_sn_f5564 <int>, new_sn_f65 <int>, new_ep_m014 <int>,
## #   new_ep_m1524 <int>, new_ep_m2534 <int>, new_ep_m3544 <int>,
```

```
## #   new_ep_m4554 <int>, new_ep_m5564 <int>, new_ep_m65 <int>,
## #   new_ep_f014 <int>, new_ep_f1524 <int>, new_ep_f2534 <int>,
## #   new_ep_f3544 <int>, new_ep_f4554 <int>, new_ep_f5564 <int>,
## #   new_ep_f65 <int>, newrel_m014 <int>, newrel_m1524 <int>,
## #   newrel_m2534 <int>, newrel_m3544 <int>, newrel_m4554 <int>,
## #   newrel_m5564 <int>, newrel_m65 <int>, newrel_f014 <int>,
## #   newrel_f1524 <int>, newrel_f2534 <int>, newrel_f3544 <int>,
## #   newrel_f4554 <int>, newrel_f5564 <int>, newrel_f65 <int>
```

We use the function `gather()` to get rid of several columns leaving out NAs.

```
who1 <- who %>%
  gather(new_sp_m014:newrel_f65, key = "key", value = "cases", na.rm = TRUE)
head(who1)
```

```
## # A tibble: 6 x 6
##   country     iso2  iso3   year key         cases
##   <chr>       <chr> <chr> <int> <chr>       <int>
## 1 Afghanistan AF    AFG    1997 new_sp_m014     0
## 2 Afghanistan AF    AFG    1998 new_sp_m014    30
## 3 Afghanistan AF    AFG    1999 new_sp_m014     8
## 4 Afghanistan AF    AFG    2000 new_sp_m014    52
## 5 Afghanistan AF    AFG    2001 new_sp_m014   129
## 6 Afghanistan AF    AFG    2002 new_sp_m014    90
```

We can count the keys we just created to get a sense we have done it right.

```
who1 %>%
  count(key)
```

```
## # A tibble: 56 x 2
##    key              n
##    <chr>        <int>
##  1 new_ep_f014   1032
##  2 new_ep_f1524  1021
##  3 new_ep_f2534  1021
##  4 new_ep_f3544  1021
##  5 new_ep_f4554  1017
##  6 new_ep_f5564  1017
##  7 new_ep_f65    1014
##  8 new_ep_m014   1038
##  9 new_ep_m1524  1026
## 10 new_ep_m2534  1020
## # ... with 46 more rows
```

Correcting some inconsistency concerning the values of the columns `key`.

```
who2 <- who1 %>%
  mutate(key = stringr::str_replace(key, "newrel", "new_rel"))
head(who2)
```

```
## # A tibble: 6 x 6
##   country     iso2  iso3   year key         cases
##   <chr>       <chr> <chr> <int> <chr>       <int>
## 1 Afghanistan AF    AFG    1997 new_sp_m014     0
## 2 Afghanistan AF    AFG    1998 new_sp_m014    30
## 3 Afghanistan AF    AFG    1999 new_sp_m014     8
## 4 Afghanistan AF    AFG    2000 new_sp_m014    52
## 5 Afghanistan AF    AFG    2001 new_sp_m014   129
## 6 Afghanistan AF    AFG    2002 new_sp_m014    90
```

We split values and create new columns.

```
who3 <- who2 %>%
  separate(key, c("new", "type", "sexage"), sep = "_")
head(who3)
```

```
## # A tibble: 6 x 8
##   country     iso2  iso3   year new   type  sexage cases
##   <chr>       <chr> <chr> <int> <chr> <chr> <chr>  <int>
## 1 Afghanistan AF    AFG    1997 new   sp    m014       0
## 2 Afghanistan AF    AFG    1998 new   sp    m014      30
## 3 Afghanistan AF    AFG    1999 new   sp    m014       8
## 4 Afghanistan AF    AFG    2000 new   sp    m014      52
## 5 Afghanistan AF    AFG    2001 new   sp    m014     129
## 6 Afghanistan AF    AFG    2002 new   sp    m014      90
```

We drop columns we do not use or redundant.

```
who4 <- who3 %>%
  select(-new, -iso2, -iso3)
head(who4)
```

```
## # A tibble: 6 x 5
##   country      year type  sexage cases
##   <chr>       <int> <chr> <chr>  <int>
## 1 Afghanistan  1997 sp    m014       0
## 2 Afghanistan  1998 sp    m014      30
## 3 Afghanistan  1999 sp    m014       8
## 4 Afghanistan  2000 sp    m014      52
## 5 Afghanistan  2001 sp    m014     129
## 6 Afghanistan  2002 sp    m014      90
```

You may have noticed that `sexage` is still not a standalone columns. Let's split that!

```
who5 <- who4 %>%
  separate(sexage, c("sex", "age"), sep = 1)
head(who5)
```

```
## # A tibble: 6 x 6
##   country      year type  sex   age   cases
##   <chr>       <int> <chr> <chr> <chr> <int>
## 1 Afghanistan  1997 sp    m     014       0
```

```
## 2 Afghanistan  1998 sp    m     014      30
## 3 Afghanistan  1999 sp    m     014       8
## 4 Afghanistan  2000 sp    m     014      52
## 5 Afghanistan  2001 sp    m     014     129
## 6 Afghanistan  2002 sp    m     014      90
```

Putting it all together in a huge pipe.

```
who %>%
  gather(key, value, new_sp_m014:newrel_f65, na.rm = TRUE) %>%
  mutate(key = stringr::str_replace(key, "newrel", "new_rel")) %>%
  separate(key, c("new", "var", "sexage")) %>%
  select(-new, -iso2, -iso3) %>%
  separate(sexage, c("sex", "age"), sep = 1)
```

```
## # A tibble: 76,046 x 6
##    country      year var   sex   age   value
##    <chr>       <int> <chr> <chr> <chr> <int>
##  1 Afghanistan  1997 sp    m     014       0
##  2 Afghanistan  1998 sp    m     014      30
##  3 Afghanistan  1999 sp    m     014       8
##  4 Afghanistan  2000 sp    m     014      52
##  5 Afghanistan  2001 sp    m     014     129
##  6 Afghanistan  2002 sp    m     014      90
##  7 Afghanistan  2003 sp    m     014     127
##  8 Afghanistan  2004 sp    m     014     139
##  9 Afghanistan  2005 sp    m     014     151
## 10 Afghanistan  2006 sp    m     014     193
## # ... with 76,036 more rows
```

Now, we have a nice tidy table. We can start the analysis!

Exercise 6:

I am supposed to tidy the following dataset.

```
df <- tibble(name = c("A123", "B456"),
             age = c(30, 60),
             answer1 = c(0, 1),
             answer2 = c(1,1),
             answer3 = c(1,0),
             answer4 = c(0,0))
head(df)
```

```
## # A tibble: 2 x 6
##   name    age answer1 answer2 answer3 answer4
##   <chr> <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
## 1 A123     30       0       1       1       0
## 2 B456     60       1       1       0       0
```

First, I realize that they are the `answerN` columns that should be modified. If I want to add more answer types(?) as dimensions to this table I always have to create a new column. I do not want that. I would only want to add a new row to the existing table. Accordingly, I keep `name` and `age` and add a column to assess the 'type' of the answer and the value of it.

Make it tidy with `gather()`:

```
df_tidy <- df %>%
  gather(`answer1`,
         `answer2`,
         `answer3`,
         `answer4`,
         key = "type of answer",
         value = "value of the answer")
head(df_tidy)
```

```
## # A tibble: 6 x 4
##   name    age `type of answer` `value of the answer`
##   <chr> <dbl> <chr>                             <dbl>
## 1 A123     30 answer1                               0
## 2 B456     60 answer1                               1
## 3 A123     30 answer2                               1
## 4 B456     60 answer2                               1
## 5 A123     30 answer3                               1
## 6 B456     60 answer3                               0
```