MATH2349 Data Wrangling

Assignment 1

Rinaldo Gagiano ~ S3870806

Setup

Installing and loading of necassary packages to reproduce this report:

```
library(readr) # Useful for importing data
library(tidyverse) # Useful for data frame manipulation
```

```
## -- Attaching packages ---------------------------------------------------------------------------
## v ggplot2 3.3.2     v dplyr   1.0.1
## v tibble  2.1.3     v stringr 1.4.0
## v tidyr   1.1.1     v forcats 0.4.0
## v purrr   0.3.3
## -- Conflicts ------------------------------------------------------------------------------------
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

Data Description

The dataset 'price-of-weed' comes from the user 'frankbi' hosted on GitHub. It was taken from the URL: https://github.com/frankbi/price-of-weed/raw/master/data/weedprices01012014.csv. The dataset contains weed quality pricing, in US dollars, and the number of weed quality reports within each state of the USA.

The 'price-of-weed' dataset consists of 7 variables highlighted below:

- 'State' variable consisting of the names of each state located within the USA.
- 'HighQ' variable denoting the average price, in US Dollars, of high-quality weed within the state.
- 'HighQN' variable totalling the quantity of high-quality weed price reports made within the state.
- 'MedQ' variable denoting the average price, in US Dollars, of medium-quality weed within the state.
- 'MedQn' variable totalling the quantity of medium-quality weed price reports made within the state.
- 'LowQ' variable denoting the average price, in US Dollars, of low-quality weed within the state.
- 'LowQN' variable totalling the quantity of low-quality weed price reports made within the state.

Read/Import Data

The function 'read_csv()' from the 'readr' package is used to read the dataset. The dataset's URL is inserted into the function's input surrounded by quotation marks. This is then assigned to the arbitrary variable name 'df' as shown in the code below:

```
df<-read_csv("https://github.com/frankbi/price-of-weed/raw/master/data/weedprices01012014.csv")
```

The function 'head()' is used to produce a small snapshot output of the data frame, consisting of all the columns but not all the rows. The variable 'df' is inserted in to the function's input as seen below:

```
head(df)
```

```
## # A tibble: 6 x 7
##   State      HighQ   HighQN MedQ    MedQN LowQ    LowQN
##   <chr>      <chr>    <dbl> <chr>   <dbl> <chr>   <dbl>
## 1 Alabama    $339.06   1042 $198.64   933 $149.49   123
## 2 Alaska     $288.75    252 $260.6    297 $388.58    26
## 3 Arizona    $303.31   1941 $209.35  1625 $189.45   222
## 4 Arkansas   $361.85    576 $185.62   544 $125.87   112
## 5 California $248.78  12096 $193.56 12812 $192.92   778
## 6 Colorado   $236.31   2161 $195.29  1728 $213.5    128
```

Inspect and Understand

Dimensions

The dimensions of the data frame can be checked using the 'dim()' function. Inserting 'df' into the input of the function gives us the output below:

```
dim(df)
```

```
## [1] 51  7
```

As we can see here the dimensions of our data frame consists of 51 observations and 7 variables.

Data Types

Variable Data Type Check

Each variable in our data frame is in a certain data type. To check the data types of our variables, we can use the 'mode()' function. To check the data type of variable 'State' we specify in the functions input 'df$State'. The dollar sign between our data frame name and variable name specifies which variable we wish to use. This is produced below:

```
mode(df$State)
```

```
## [1] "character"
```

To save time and space, we can use the function 'sapply()' to apply a certain function to an element the same number of times as the length of the element. For this function we will input our data frame 'df' and the function 'mode()' to obtain the data types of all variables within our data frame. This can be seen below:

```
sapply(df,mode)
```

```
##       State       HighQ      HighQN        MedQ       MedQN        LowQ
## "character" "character"   "numeric" "character"   "numeric" "character"
##       LowQN
##   "numeric"
```

Above we can see each variables' data type. According to this 'State', 'HighQ', 'MedQ', and 'LowQ' are all character-based variables. 'HighQN', 'MedQN', and 'LowQN' are all numeric based variables. This indicates that variables 'HighQ', 'MedQ', and 'LowQ' are all in the wrong format because we wish them to be numerical values, as they are price in US dollars, rather than a character string. This can be corrected as shown below.

Variable Data Type Change

To change the data type of a particular variable from character-based to numeric-based, we use the function 'as.numeric()'. The code for changing the data type of variable 'HighQN' to numeric is produced below:

```
as.numeric(df$HighQ)
```

```
## Warning: NAs introduced by coercion
```

```
##  [1] NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA
## [26] NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA
## [51] NA
```

Our code above produced nothing but 'NA'. This indicates something is not right. By checking a sample of our variable, using the function 'head()' again, we can pick up whats wrong:

```
head(df$HighQ)
```

```
## [1] "$339.06" "$288.75" "$303.31" "$361.85" "$248.78" "$236.31"
```

As we can see from our sample, each value in our variable 'HighQ' starts with the special character '$'. This is the reason our variable was initially assigned to the character-based data type. In order for us to change our data type to 'numeric', we must first remove all non-numeric values from each value within the variable. To do this we use the function 'gsub()' which replaces a certain input with a new defined input. The inputs we will give the function will be "\\$","", and the desired variable's name. The first input denotes us choosing the dollar sign. The double slashes in front indicate us escaping or 'breaking' the special character as the dollar sign has a unique purpose when defining what text to choose for replacement. We do not need to go into those details as it will not be necessary for this document. The second input is just an empty replacement as we wish to replace the dollar sign with nothing. The third input is the desired variable we wish to change. To alter the variable, each one must be reassigned to itself with the appropriate functional changes as seen above. This code can be seen below:

```
df$HighQ <- gsub("\\$", "", df$HighQ)
df$MedQ <- gsub("\\$", "", df$MedQ)
df$LowQ <- gsub("\\$", "", df$LowQ)
```

Above we can see that the three variables that were changed. In order to check that the above code was successful, we can reuse the 'head()' function discussed above:

```
head(df)
```

```
## # A tibble: 6 x 7
##   State      HighQ  HighQN MedQ   MedQN LowQ   LowQN
##   <chr>      <chr>   <dbl> <chr>  <dbl> <chr>  <dbl>
## 1 Alabama    339.06   1042 198.64   933 149.49   123
## 2 Alaska     288.75    252 260.6    297 388.58    26
## 3 Arizona    303.31   1941 209.35  1625 189.45   222
## 4 Arkansas   361.85    576 185.62   544 125.87   112
## 5 California 248.78  12096 193.56 12812 192.92   778
## 6 Colorado   236.31   2161 195.29  1728 213.5    128
```

From the table above, we can see that all three modified variables are in the correct format. Now we can return to using the 'as.numeric()' function discussed above. Remember that we must reassign the data frame variables in order to change them, as seen in the code below:

```
df$HighQ <- as.numeric(df$HighQ)
df$MedQ <- as.numeric(df$MedQ)
df$LowQ <- as.numeric(df$LowQ)
```

To check that the changes made were correct we can revisit our 'sapply()' and 'mode()' functions:

```
sapply(df,mode)
```

```
##       State        HighQ      HighQN        MedQ       MedQN        LowQ
## "character"   "numeric"   "numeric"   "numeric"   "numeric"   "numeric"
##       LowQN
##   "numeric"
```

Excellent, we can now see that all our data frame variables are in the correct data type.

Factor and Levels

Factors are useful for representing categorical date. Within our data frame, none of our variables would benefit from being categorised so no factors or levels will be used.

Column Name Corrections

Checking of Column Names

To check the current column names of our data frame, we can use the 'colnames()' function, including 'df' as the functions input:

```
colnames(df)
```

```
## [1] "State"  "HighQ"  "HighQN" "MedQ"   "MedQN"  "LowQ"   "LowQN"
```

From the information above we can get a general idea of what the columns represent. There is room for improvement though.

Changing of Column Names

Changing of column names can be done using the 'rename()' function. This function's inputs take 'df', the data frame containing the columns, and 'new_name=old_name'. The second input can be repeated for as many columns that need to be changed. Reassigning the data frame will save the new column names as seen below:

```
df <- rename(df,HQ_Price=HighQ,
             HQ_Reports=HighQN,
             MQ_Price=MedQ,
             MQ_Reports=MedQN,
             LQ_Price=LowQ,
             LQ_Reports=LowQN)
```

Let's recheck the data frame's column names using the 'colnames()' function:

```
colnames(df)
```

```
## [1] "State"      "HQ_Price"   "HQ_Reports" "MQ_Price"   "MQ_Reports"
## [6] "LQ_Price"   "LQ_Reports"
```

Great, we can see that all the desired changes were made and the column names give us a better indication of what the data is conveying.

Final Check

Let's do one last final check of our modified data frame to make sure everything is in order. We can do this by simply calling the variable name assigned to our data frame:

```
df
```

```
## # A tibble: 51 x 7
##    State              HQ_Price HQ_Reports MQ_Price MQ_Reports LQ_Price LQ_Reports
##    <chr>                 <dbl>      <dbl>    <dbl>      <dbl>    <dbl>      <dbl>
##  1 Alabama              339.06       1042   198.64        933   149.49        123
##  2 Alaska               288.75        252   260.6         297   388.58         26
##  3 Arizona              303.31       1941   209.35       1625   189.45        222
##  4 Arkansas             361.85        576   185.62        544   125.87        112
##  5 California           248.78      12096   193.56      12812   192.92        778
##  6 Colorado             236.31       2161   195.29       1728   213.5         128
##  7 Connecticut          347.9        1294   273.97       1316   257.36         91
##  8 Delaware             373.18        347   226.25        273   199.88         34
##  9 District of Colu~    352.26        433   295.67        349   213.72         39
## 10 Florida              306.43       6506   220.03       5237   158.26        514
## # ... with 41 more rows
```

Everything seems to be in order.

Subsetting

Subsetting the data frame

In order to subset the data frame, we must create a new variable to store the subset. I have chosen the name 'subDf' but this is arbitrary. We wish to only take the first 10 values from each variable in the previous data frame. To do this we can revisit our function 'head()' with a slight addition to the inputs. The first input will remain 'df', as this is the data frame we wich to subset from, but our second input will be an integer indicating how many observations we wish to display. In this case, we will use '10' as shown below:

```
subDf <- head(df,10)
```

To check this was done correctly we call our newly created data frame with the code below:

```
subDf
```

```
## # A tibble: 10 x 7
##    State             HQ_Price HQ_Reports MQ_Price MQ_Reports LQ_Price LQ_Reports
##    <chr>                <dbl>      <dbl>    <dbl>      <dbl>    <dbl>      <dbl>
##  1 Alabama              339.06       1042   198.64        933   149.49        123
##  2 Alaska               288.75        252   260.6         297   388.58         26
##  3 Arizona              303.31       1941   209.35       1625   189.45        222
##  4 Arkansas             361.85        576   185.62        544   125.87        112
##  5 California           248.78      12096   193.56      12812   192.92        778
##  6 Colorado             236.31       2161   195.29       1728   213.5         128
##  7 Connecticut          347.9        1294   273.97       1316   257.36         91
##  8 Delaware             373.18        347   226.25        273   199.88         34
##  9 District of Colu~    352.26        433   295.67        349   213.72         39
## 10 Florida              306.43       6506   220.03       5237   158.26        514
```

Creating the Matrix

Initial Matrix

To convert our subset into a matrix we can use the function 'data.matrix()'. This function will take the input of the desired data frame we wish to convert. In our case, it is 'subDf'. I have also assigned this matrix to a new variable name 'matrixDf' as seen below:

```
matrixDf <- data.matrix(subDf)
```

```
## Warning in data.matrix(subDf): NAs introduced by coercion
```

To reveal this matrix we can call its name:

```
matrixDf
```

```
##       State HQ_Price HQ_Reports MQ_Price MQ_Reports LQ_Price LQ_Reports
##  [1,]    NA   339.06       1042   198.64        933   149.49        123
##  [2,]    NA   288.75        252   260.60        297   388.58         26
##  [3,]    NA   303.31       1941   209.35       1625   189.45        222
##  [4,]    NA   361.85        576   185.62        544   125.87        112
##  [5,]    NA   248.78      12096   193.56      12812   192.92        778
##  [6,]    NA   236.31       2161   195.29       1728   213.50        128
##  [7,]    NA   347.90       1294   273.97       1316   257.36         91
##  [8,]    NA   373.18        347   226.25        273   199.88         34
##  [9,]    NA   352.26        433   295.67        349   213.72         39
## [10,]    NA   306.43       6506   220.03       5237   158.26        514
```

As we see above our matrix contains an error in our 'State' column. The reason for this error will be discussed later. For now, let's just fix this.

Fixing Matrix

Because our 'State' column was causing our matrix issues, we will create a new matrix, over the top of our old one, excluding the 'state' column. To do this we follow the 'data.matrix()' function explanation above but we must alter our input. Since we want all 7 columns except for the first, we index our data frame from column 2 through 7 using the code "[2:7]". This can be previewed below:

```
matrixDf <- data.matrix(subDf[2:7])
```

To ensure this worked correctly, let's call our new matrix:

```
matrixDf
```

```
##       HQ_Price HQ_Reports MQ_Price MQ_Reports LQ_Price LQ_Reports
##  [1,]   339.06       1042   198.64        933   149.49        123
##  [2,]   288.75        252   260.60        297   388.58         26
##  [3,]   303.31       1941   209.35       1625   189.45        222
##  [4,]   361.85        576   185.62        544   125.87        112
##  [5,]   248.78      12096   193.56      12812   192.92        778
##  [6,]   236.31       2161   195.29       1728   213.50        128
##  [7,]   347.90       1294   273.97       1316   257.36         91
##  [8,]   373.18        347   226.25        273   199.88         34
##  [9,]   352.26        433   295.67        349   213.72         39
## [10,]   306.43       6506   220.03       5237   158.26        514
```

We can now see that we don't have any errors but the state column is missing. To rectify this we can change our matrix row names to the names of our 'State' column.

Matrix Row Name Change

Since we wish to change our matrix row names to the state names from our data frames column 'State', we can use the function 'rownames()'. This function alters data on its own so there is no need to reassign any variables. The input for the function will be our matrix name 'matrixDf'. In order to give it the right row names, we will assign the row names by inputting 'subDf$State' as seen below:

```
rownames(matrixDf) <- subDf$State
```

To finalise everything lets call up our matrix:

```
matrixDf
```

```
##                      HQ_Price HQ_Reports MQ_Price MQ_Reports LQ_Price
## Alabama                339.06       1042   198.64        933   149.49
## Alaska                 288.75        252   260.60        297   388.58
## Arizona                303.31       1941   209.35       1625   189.45
## Arkansas               361.85        576   185.62        544   125.87
## California             248.78      12096   193.56      12812   192.92
## Colorado               236.31       2161   195.29       1728   213.50
## Connecticut            347.90       1294   273.97       1316   257.36
## Delaware               373.18        347   226.25        273   199.88
## District of Columbia   352.26        433   295.67        349   213.72
## Florida                306.43       6506   220.03       5237   158.26
##                      LQ_Reports
## Alabama                     123
## Alaska                       26
## Arizona                     222
## Arkansas                    112
## California                  778
## Colorado                    128
```

```
## Connecticut                  91
## Delaware                     34
## District of Columbia         39
## Florida                     514
```

Excellent! We can see our matrix is fixed.

Matrix Structure Check

Our matrix has a particular structure. To check this structure we can rely on our 'mode()' function used above. The input for this function will be 'matrixDf':

```
mode(matrixDf)
```

```
## [1] "numeric"
```

As we can see the structure of our matrix is 'numeric'. This is due to the function 'data.matrix()', as this function coverts all variables within the data frame to 'numeric' data type, before binding together all columns of the matrix. The function converting all variables into the datatype 'numeric' is the reason we had 'NA's as the outcome of our initial matrix for the 'State' column. The 'State' column was in the datatype 'character' and as we have seen above, converting elements into a 'numeric' datatype need to be in the correct format. E.G. '345.67' -> 345.67, 'Arkansas' -> NA

Create a new Data Frame

Vector Creation

To create our data frame with 2 variables and 10 observations, we must first create two vectors. Our first vector will be our ordinal variable called 'chilli_colour', our second vector will be our integer variable called 'chilli_length_cm'. Note then when assigning the new variables, each one has a 'c' infront of the list, to indicate that it is a vector, as seen here:

```
chilli_colour <- c('Red','Yellow','Red','Green','Green','Orange','Yellow',
                   'Green','Red','Orange')
chilli_length_cm <- c(3,8,2,12,10,5,9,15,2,6)
```

Seen above are our two vectors. The vector 'chilli_colour', indicates the observed colour of each respective chilli. The vector 'chilli_length_cm', indicates the length of the respective chilli in centimetres. Now that we have our two vectors we can move onto creating our data frame.

Data Frame Creation

The forming of our data frame will be taken care of with the function 'data.frame()'. The inputs for this function will be the vectors created above. We will assign a new name for the creation of our data frame:

```
chilli_df <- data.frame(chilli_colour,chilli_length_cm)
```

Let's see if this worked by calling our new data frame:

```
chilli_df
```

```
##    chilli_colour chilli_length_cm
## 1            Red                3
## 2         Yellow                8
## 3            Red                2
## 4          Green               12
## 5          Green               10
## 6         Orange                5
## 7         Yellow                9
## 8          Green               15
## 9            Red                2
## 10        Orange                6
```

With our dataframe created, we can now run a few checks to see if everything is formatted properly.

Variable Data Type Check

Before we check the factor and levels of our data frame lets have a quick look at our variable data types, using the previously discussed function, 'sapply()' and 'mode()':

```
sapply(chilli_df,mode)
```

```
##    chilli_colour chilli_length_cm
##        "numeric"        "numeric"
```

Here we can see that both variables data types are 'numeric', but why is this seeing as our variable 'chilli_colour' are all character strings? The answer to this is in its factor.

Factor and Levels

The function 'data.frame()' converts all inputs to numeric, including character vectors. This is because of its automatic nature to factor all 'character' based variables. The variable 'chilli_colour' has been factored and

given levels based on its contents. this can seen using the 'levels()' function. As we wish to only see the levels of our 'chilli_colour' column, we will specify this in the input, as seen here:

```
levels(chilli_df$chilli_colour)
```

```
## [1] "Green"  "Orange" "Red"    "Yellow"
```

The contents above are the levels of our factored variable 'chilli_colour', but they aren't ordered correctly.

Level Ordering

To order levels of a factor we can use the 'factor()' function. Our 'chilli_colour' variable is not in the correct order and we wish to display the 'hotness' of each respective chilli, through the order 'Red','Orange','Yellow', and 'Green'. To do so we specify in the 'factor()' function's second input the 'levels=c()' argument, followed with the correct order in which we desire the levels. We must also specify which variable we wish to change, and the reassignment of the variable. This can be seen in the code below:

```
chilli_df$chilli_colour <- factor(chilli_df$chilli_colour,
                                  levels=c('Red','Orange','Yellow','Green'))
```

To check the reordering was successful, we can reuse our 'levels()' function:

```
levels(chilli_df$chilli_colour)
```

```
## [1] "Red"    "Orange" "Yellow" "Green"
```

Excellent, our levels are in order!

Variable Addition

To add another variable to our existing data frame, we must first create the vector we wish to add. The numerical vector I will create will be named 'chilli_seed_count', and it will contain the respective see count of the chillis. Remember to use the 'c' before our list to ensure it is created into a vector:

```
chilli_seed_count <- c(8,5,9,3,2,7,4,1,10,6)
```

Next, we must add this to our existing data frame. To do this we can use the function 'cbind()'. This function will take the input of the original data frame, plus the addition we wish to add. In this case, it will be our 'chilli_seed_count' vector. We must reassign our original data frame to save the changes:

```
chilli_df<-cbind(chilli_df,chilli_seed_count)
```

Let's see if this worked by calling our new modified data frame:

```
chilli_df
```

```
##    chilli_colour chilli_length_cm chilli_seed_count
## 1            Red                3                 8
## 2         Yellow                8                 5
## 3            Red                2                 9
## 4          Green               12                 3
## 5          Green               10                 2
## 6         Orange                5                 7
## 7         Yellow                9                 4
## 8          Green               15                 1
## 9            Red                2                10
## 10        Orange                6                 6
```

Perfect! Our data frame is complete.

References

Baglin, J. (2020). R Bootcamp - Course 2: Working with Data. [online] Available at: https://astral-theory-157510.appspot.com/secured/RBootcamp_Course_02.html#overview [Accessed 15 Aug. 2020].