Lecture 3

Paul Holaway, Abhi Thanvi

June 23rd, 2022

Lecture 2 Review

Example 1; Importing A Data Set Review

Before we move on to the new material, we will do a quick review of Lecture 2 content. Last time we learned how to import a data set and filter rows or columns. First, we will import a data set. Please refer to the Lecture 2 notes for the steps if you do not remember. Don't forget to copy and paste the import code. The data set you will be importing is Amtrak and NJ Transit efficiency of the North East Corridor trains.

Hint: Remember to set your working directory with the data set first.

```
Trains <- read.csv("~/Desktop/DPISu22/Data Sets/AMTK_NJT Performance 5-20.csv", stringsAsFactors=TRUE)
```

Click on the data set in your local environment to view it. You should have 98,698 rows and 13 columns. That is a **LOT** of data, which is why we will refrain from printing it.

Example 2; Column and Row Filtering

First, let's filter out columns that will not be as meaningful. Things such as the train and station IDs are not meaningful to us (we are not the railroad), so let's remove those columns.

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --
```

```
## v ggplot2 3.3.6     v purrr   0.3.4
## v tibble  3.1.7     v dplyr   1.0.9
## v tidyr   1.2.0     v stringr 1.4.0
## v readr   2.1.2     v forcats 0.5.1
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
Train = Trains %>% select(-c("train_id","to_id","from_id"))
```

This is a useful thing to do in data science. If there is not useful data, it is usually best to make another data frame that only has what you need. **DO NOT OVERWRITE THE ORIGINAL ONE!!** You never know if you have to restart the data cleaning process and overwriting the data could result in you losing that data if you do not have a backup copy.

Now let's suppose you are going to be taking Amtrak over the summer there so you just want to look at Amtrak trains. Let's filter out just the trains that are run by Amtrak.

```
TrainAMTK = Train %>% filter(type == "Amtrak")
```

Notice how the number of rows went from 98,698 to 2,082. Also notice how the scheduled time and delay minutes are blank or `NA`. This means that the data is missing or not recorded. You will encounter and learn to deal with this later. However, it is indeed frustrating because there does not appear to be useful information in this filtered version of the data. Let's re-filter the data and look at NJ Transit trains.

```
TrainNJT = Train %>% filter(type == "NJ Transit")
```

Now we have the delay times so there will be some useful information here. Notice how NJ Transit goes outside of the NE Corridor line at points so let's eliminate those. Let's also look to see if any trains are more than 20 minutes late.
Fun Fact: We can string commands together using `%>%` to save lines and show off our programming skills.

```
TrainNJTNEC = TrainNJT %>% filter(line == "Northeast Corrdr") %>% filter(delay_minutes > 20)
```

Looks like 53 times the trains were more than 20 minutes late. 53/96,616 is about 0.05% which means the train was on time or less than 20 minutes lat 99.95% of the time. That is *really* good.

Okay, now we can move onto the next portion of lecture content.

The Purpose of Data Cleaning and Intro to Experimental Design

The Purpose of Data Cleaning

In the world of statistics there is a saying. "80% of the work you do is data cleaning, 20% is actual statistics/data science." This is unfortunately true. Most of the data that you are given will not be cleaned in a way you can effectively do any kind of analysis. Usually you will have to do something to fix this. Today's lecture will be little coding and more conceptual.
Have you ever had a friend show you a table or spreadsheet and you cannot for the life of you figure out how to read the spreadsheet because it is so disorganized? That is unstructured data and is going to have to be cleaned into structured data. Usually when people are making data sets, they just throw the data in without thinking about how to organize it (unless that person is a statistician or data scientist). The point is, unstructured or disorganized data cannot be analyzed. With most data sets being unstructured or semi-structured (when there is some organization), that means we have some work to do. Below are some different kinds of data cleaning.

1. Validating Data: Here you are checking for issues with your data and making sure it is accurate and clean. Examples of cleaning are below.
2. Removing Duplicate Observations: The same observation (row of data) appearing more than once in a data set is bad because it will skew your results.
3. Removing Outliers: This is data that takes extremely larger or smaller values than what we may expect. These results can also skew results and severely too.
4. Changing/Removing Missing (Null) Values: Missing data can make doing an analysis hard. Do you remove the data or just replace the missing value with something? It really depends on the data set and your judgement.

5. Dropping irrelevant rows or columns. If the data is meaningless or determined to be highly inaccurate, it is usually best to make a new data set without these rows or columns.

I will give you an example below that I myself encountered.

Example 3; Why We Clean Data

One of the courses I took was a data management course. During that course, one of my assignments was to clean up an FBI data set from 2019 to certain specifications. The biggest issues were the data set included footnotes (which cannot be analyzed) and all numeric data included , in the numbers making it impossible to analyze them. Any numbers with a , in them will be treated as strings. I will now show you the before and after of this data.

```
FBI19_0 <- read.csv("~/Desktop/DPISu22/Data Sets/fbi-table6-cius2019-data.csv", stringsAsFactors=TRUE)
```

As you can see, it looks like an Excel spreadsheet. While that may be fine in Excel, it does **NOT** work here. To fix this, I had to eliminate the footnotes, rename the columns, eliminate the blank spaces, and adjust the city names such that they appeared without M.S.A. at the end. It was a long and challenging task. The code is provided below on how to reach the final task, but you can ignore that. Most of it is far beyond the scope of this course.

```
FBI19 <- FBI19_0[-c(1937:1941),]
FBI19 <- FBI19 %>%
          rename(`Metropolitan Statistical Area` = "Table.6") %>%
          rename(`Counties/principal cities` = "X") %>%
          rename(`Population` = "X.1") %>%
          rename(`Violent crime` = "X.2") %>%
          rename(`Murder and nonnegligent manslaughter` = "X.3") %>%
          rename(Rape1 = "X.4") %>%
          rename(Robbery = "X.5") %>%
          rename(`Aggravated assault` = "X.6") %>%
          rename(`Property crime` = "X.7") %>%
          rename(Burglary = "X.8") %>%
          rename(`Larceny- theft` = "X.9") %>%
          rename(`Motor vehicle theft` = "X.10")
FBI19 <- FBI19[-c(1:3),]
row.names(FBI19) <- NULL
#Creation of 1st and 2nd Columns
X1 <- FBI19 %>%
          filter(`Metropolitan Statistical Area` != "") %>%
          select(`Metropolitan Statistical Area`,`Population`)
X1$Population <- str_remove_all(X1$Population,",")
X1$Population <- as.numeric(X1$Population)
#Creation of 3rd - 11th Columns
X2 <- FBI19 %>%
          filter(`Counties/principal cities` == "Rate per 100,000 inhabitants") %>%
          select(-c(`Metropolitan Statistical Area`,`Counties/principal cities`,`Population`))
X2$`Violent crime` <- str_remove_all(X2$`Violent crime`,",")
X2$`Murder and nonnegligent manslaughter` <- str_remove_all(X2$`Murder and nonnegligent manslaughter`,
X2$Rape1 <- str_remove_all(X2$Rape1, ",")
X2$Robbery <- str_remove_all(X2$Robbery, ",")
X2$`Aggravated assault` <- str_remove_all(X2$`Aggravated assault`, ",")
```

```
X2$`Property crime` <- str_remove_all(X2$`Property crime`, ",")
X2$Burglary <- str_remove_all(X2$Burglary, ",")
X2$`Larceny- theft` <- str_remove_all(X2$`Larceny- theft`, ",")
X2$`Motor vehicle theft` <- str_remove_all(X2$`Motor vehicle theft`, ",")
X2$`Violent crime` <- as.numeric(X2$`Violent crime`)
X2$`Murder and nonnegligent manslaughter` <- as.numeric(X2$`Murder and nonnegligent manslaughter`)
X2$Rape1 <- as.numeric(X2$Rape1)
X2$Robbery <- as.numeric(X2$Robbery)
X2$`Aggravated assault` <- as.numeric(X2$`Aggravated assault`)
X2$`Property crime` <- as.numeric(X2$`Property crime`)
X2$Burglary <- as.numeric(X2$Burglary)
X2$`Larceny- theft` <- as.numeric(X2$`Larceny- theft`)
X2$`Motor vehicle theft` <- as.numeric(X2$`Motor vehicle theft`)
FBI19_10 <- cbind(X1,X2)
FBI19_11 <- FBI19_10 %>%
              rename(Violent_crime_ratePer100K = "Violent crime") %>%
              rename(Murder_ratePer100K = "Murder and nonnegligent manslaughter") %>%
              rename(Rape_ratePer100K = "Rape1") %>%
              rename(Robbery_ratePer100K = "Robbery") %>%
              rename(Aggravated_assault_ratePer100K = "Aggravated assault") %>%
              rename(Property_crime_ratePer100K = "Property crime") %>%
              rename(Burglary_ratePer100K = "Burglary") %>%
              rename(Larceny_theft_ratePer100K = "Larceny- theft") %>%
              rename(Motor_vehicle_theft_ratePer100K = "Motor vehicle theft")
FBI19_12 = FBI19_11
X3 <- str_split(FBI19_11$`Metropolitan Statistical Area`, ",", simplify = TRUE)
X3 <- data.frame(X3)
X3$X1 <- str_remove_all(X3$X1, " ")
X3 <- X3[,-3]
X4 <- str_split(X3$X2, " ", simplify = TRUE)
X4 <- data.frame(X4)
X4 <- X4[,-c(1,3,4)]
X4 <- data.frame(X4)
X4$X4 <- str_replace_all(X4$X4, "Puerto", "PR")
FBI19_12$`Metropolitan Statistical Area` <- str_c(X3$X1, ",", X4$X4)
head(FBI19_12,10)
```

```
##    Metropolitan Statistical Area Population Violent_crime_ratePer100K
## 1                     Abilene,TX     171125                     317.3
## 2                       Akron,OH     703784                     371.7
## 3                      Albany,GA     149257                     717.6
## 4              Albany-Lebanon,OR     128105                     136.6
## 5       Albany-Schenectady-Troy,NY     879862                     257.5
## 6                 Albuquerque,NM     918114                    1043.4
## 7                  Alexandria,LA     152025                     849.2
## 8                     Altoona,PA     121762                     333.4
## 9                    Amarillo,TX     266054                     595.4
## 10                       Ames,IA     124947                     199.3
##    Murder_ratePer100K Rape_ratePer100K Robbery_ratePer100K
## 1                 3.5             61.4                42.1
## 2                 4.7             45.2                64.7
## 3                 8.0             44.2               135.3
## 4                 4.7             27.3                18.7
## 5                 1.6            39.8               54.9
## 6                10.1            64.0              194.0
## 7                10.5            46.7              108.5
## 8                 0.8            91.2               27.1
## 9                 6.0            72.9               96.6
## 10                1.6            51.2               17.6
##    Aggravated_assault_ratePer100K Property_crime_ratePer100K
## 1                           210.4                     2105.5
## 2                           257.2                     2052.5
## 3                           530.0                     3388.1
## 4                            85.9                     2141.2
## 5                           161.3                     1746.5
## 6                           775.3                         NA
## 7                           683.4                     4538.7
## 8                           214.4                     1240.1
## 9                           419.8                     3224.5
## 10                          128.9                     1396.6
##    Burglary_ratePer100K Larceny_theft_ratePer100K
## 1                 447.0                    1517.0
## 2                 393.6                    1531.9
## 3                 719.6                    2445.4
## 4                 267.0                    1687.7
## 5                 198.1                    1476.1
## 6                    NA                    2633.7
## 7                1106.4                    3160.7
## 8                 222.6                     955.1
## 9                 620.2                    2203.7
## 10                204.9                    1105.3
##    Motor_vehicle_theft_ratePer100K
## 1                            141.4
## 2                            127.0
## 3                            223.1
## 4                            186.6
## 5                             72.3
## 6                            674.0
## 7                            271.7
## 8                             62.4
## 9                            400.7
## 10                            86.4
```

Holy cow!!! That is a **LOT** of code there. I bet you can imagine this took me a *long* time to do. Do not worry though, most of this is far beyond the scope of this course.

Renaming Columns

There are two things here that are within the scope of this course. The first is renaming a column. This is a simple thing to do using the `rename()` function. Sometimes the column name does not feel intuitive to us, so we would rename it to make it easier on ourselves when we do analysis. Let's go back to our `Train` data.

Example 4; Renaming Columns

In the `Train` data, we see that there is a variable called `stop_sequence`. Most railroad people know this jargon, but some may not. You may understand what it means, but something else would make more sense

to you, say `stop_order`. Let's rename the `stop_sequence` column to `stop_order`.
The syntax for this is `Data = Data %>% rename(NewName = "OldName")`.

```
Train = Train %>% rename(stop_order = "stop_sequence")
head(Train,10)
```

```
##          date stop_order                from                  to
## 1  2020-05-01          1 Newark Penn Station Newark Penn Station
## 2  2020-05-01          2 Newark Penn Station               Union
## 3  2020-05-01          3               Union        Roselle Park
## 4  2020-05-01          4        Roselle Park            Cranford
## 5  2020-05-01          5            Cranford          Westfield
## 6  2020-05-01          6           Westfield             Fanwood
## 7  2020-05-01          7             Fanwood          Netherwood
## 8  2020-05-01          8          Netherwood          Plainfield
## 9  2020-05-01          9          Plainfield            Dunellen
## 10 2020-05-01         10            Dunellen         Bound Brook
##         scheduled_time         actual_time delay_minutes   status
## 1  2020-05-01 23:38:00 2020-05-01 23:40:09    2.15000000 departed
## 2  2020-05-01 23:47:00 2020-05-01 23:47:01    0.01666667 departed
## 3  2020-05-01 23:50:00 2020-05-01 23:51:04    1.06666667 departed
## 4  2020-05-01 23:55:00 2020-05-01 23:55:31    0.51666667 departed
## 5  2020-05-01 23:59:00 2020-05-01 23:59:01    0.01666667 departed
## 6  2020-05-02 00:04:00 2020-05-02 00:03:21    0.00000000 departed
## 7  2020-05-02 00:07:00 2020-05-02 00:07:10    0.16666667 departed
## 8  2020-05-02 00:11:00 2020-05-02 00:09:05    0.00000000 departed
## 9  2020-05-02 00:16:00 2020-05-02 00:13:01    0.00000000 departed
## 10 2020-05-02 00:21:00 2020-05-02 00:19:07    0.00000000 departed
##              line       type
## 1  Raritan Valley NJ Transit
## 2  Raritan Valley NJ Transit
## 3  Raritan Valley NJ Transit
## 4  Raritan Valley NJ Transit
## 5  Raritan Valley NJ Transit
## 6  Raritan Valley NJ Transit
## 7  Raritan Valley NJ Transit
## 8  Raritan Valley NJ Transit
## 9  Raritan Valley NJ Transit
## 10 Raritan Valley NJ Transit
```

Creating Columns

Now we will move onto creating new columns. This is also a simple thing to do using the `mutate()` function. Sometimes we wish for data to be in a different format than what it is in now, but we do not want to overwrite the old data. This is when creating a new column comes in handy. The two most common uses of creating a new column are…

- If a new variable is a mathematical expression of current variables.
- If we are converting units (minutes to hours).

Example 5; Creating Columns

In the `Train` data, we see that there is a variable called `delay_minutes`. However, what if we wanted the delay time in hours instead? Let's convert the delay time from minutes to hours. The syntax for this is `Data = Data %>% mutate(NewColumnName = Expression)`.

```
Train = Train %>% mutate(delay_hours = delay_minutes/60)
head(Train, 10)
```

```
##          date stop_order               from                 to
## 1  2020-05-01          1 Newark Penn Station Newark Penn Station
## 2  2020-05-01          2 Newark Penn Station              Union
## 3  2020-05-01          3               Union       Roselle Park
## 4  2020-05-01          4        Roselle Park           Cranford
## 5  2020-05-01          5            Cranford          Westfield
## 6  2020-05-01          6           Westfield            Fanwood
## 7  2020-05-01          7             Fanwood         Netherwood
## 8  2020-05-01          8          Netherwood         Plainfield
## 9  2020-05-01          9          Plainfield           Dunellen
## 10 2020-05-01         10            Dunellen        Bound Brook
##         scheduled_time         actual_time delay_minutes   status
## 1  2020-05-01 23:38:00 2020-05-01 23:40:09    2.15000000 departed
## 2  2020-05-01 23:47:00 2020-05-01 23:47:01    0.01666667 departed
## 3  2020-05-01 23:50:00 2020-05-01 23:51:04    1.06666667 departed
## 4  2020-05-01 23:55:00 2020-05-01 23:55:31    0.51666667 departed
## 5  2020-05-01 23:59:00 2020-05-01 23:59:01    0.01666667 departed
## 6  2020-05-02 00:04:00 2020-05-02 00:03:21    0.00000000 departed
## 7  2020-05-02 00:07:00 2020-05-02 00:07:10    0.16666667 departed
## 8  2020-05-02 00:11:00 2020-05-02 00:09:05    0.00000000 departed
## 9  2020-05-02 00:16:00 2020-05-02 00:13:01    0.00000000 departed
## 10 2020-05-02 00:21:00 2020-05-02 00:19:07    0.00000000 departed
##               line       type  delay_hours
## 1  Raritan Valley NJ Transit 0.0358333333
## 2  Raritan Valley NJ Transit 0.0002777778
## 3  Raritan Valley NJ Transit 0.0177777778
## 4  Raritan Valley NJ Transit 0.0086111111
## 5  Raritan Valley NJ Transit 0.0002777778
## 6  Raritan Valley NJ Transit 0.0000000000
## 7  Raritan Valley NJ Transit 0.0027777778
## 8  Raritan Valley NJ Transit 0.0000000000
## 9  Raritan Valley NJ Transit 0.0000000000
## 10 Raritan Valley NJ Transit 0.0000000000
```

Intro to Experimental Design

In the world of statistics/data science, we general follow a four step sequence for doing any kind of statistical analysis.

1. Claim/Hypothesis: Any statement regarding the unknown population parameter you are studying (Maximum, Minimum, Average/Mean, etc.).
2. Experiment: Design the study

- Collecting the data set

- Designing the study

3. Likelihood: Analyzing the data set
4. Conclusion: Inference

You can already do number 1. However, now we need to teach you how to do number 2-4. We will be introducing number 2 today and finishing on Monday. 3-4 will be taught to you later in the course. Many universities will have an entire course on experimental design and multiple courses for analysis and inference. It is a broad scope, but for now we will just give you a small taste of what is out there.

Anecdotal vs. Available Data

- Anecdotal Data: Represents individual cases that often come to our attention because they are striking in some way.
- Available Data: Data that were produced in the past for some other purpose, but that may help answer a present question inexpensively.

When something catches your attention, that will be anecdotal data. Yes, it is indeed data because you observed it. That leads to a question, which is number 1 in the sequence. How you have to go about collecting the data. Data collection is done via sampling. We will go over some basic sampling ideas in the next lecture and go over how to do the basics in `RStudio`. We will go over more complex sampling later in the course. We will also go over basic experimental design next time too.

End of Lecture 3 Notes