Data Wrangling using R Rstudio

Bernardo Vimpi

9/27/2020

In this project we use R packages and library to perform Data Wrangling which is an important step in Data Science life cycle process.

Packages

```
library(dplyr, warn.conflicts = FALSE)
library(ggplot2)
library(nycflights13)
library(tinytex)
```

#Lets explore the flight dataset and concentrate on flights from NYC to Portland, Oregon.

```
portland_flights <- flights %>%
  filter(dest == "PDX")
View(portland_flights)
```

Filter the flights that departed from JFK to Burlington, Vermont, or Seattle, Washington during the months of October, November or December.

```
btv_sea_flights_fall <- flights%>%
  filter(origin== "JFK" & (dest == "BTV" | dest == "SEA") & month >=10)
View(btv_sea_flights_fall)
```

For the criteria above, lets not select the flights whose destinations was not Burlington, Vermont or Seattle, Washington.

```
not_BTV_SEA <- flights %>%
  filter(!(dest== "BTV" | dest == "SEA"))
View(not_BTV_SEA)
```

Lets concatinate the data to include multiple airports instead of using the | operator multiple times.

```
many_airports <- flights%>%
  filter(dest %in% c("SEA", "SFO", "PDX", "BTV", "BDL"))
View(many_airports)
```

##(LC3.1) What's another way of using the "not" operator ! to filter only the rows that are not going to Burlington, VT nor Seattle, WA in the flights data frame? Test this out using the previous code.

```
not2_BTV_SAW <-flights%>%
  filter( !dest == "BTV", !dest =="SEA")
View(not2_BTV_SAW)
```

```
not2_BTV_SAW <-flights%>%
  filter(dest != "BTV", dest !="SEA")
View(not2_BTV_SAW)
```

Summarize function.

```
summary_temp <- weather %>%
  summarize(mean = mean(temp, na.rm= TRUE),
            std_dev = sd(temp, na.rm = "TRUE"))
summary_temp
```

```
## # A tibble: 1 x 2
##    mean std_dev
##   <dbl>   <dbl>
## 1  55.3    17.8
```

(LC3.2) Say a doctor is studying the effect of smoking on lung cancer for a large number of patients who have records measured at five years intervals. She notices that a larger number of patients have missing data points because the patient has died, so she chooses to ignore these patients in her analysis. What is wrong with this doctor's approach?

Answer: Generally speaking, the idea of deleting missing values could work in some cases but not in others. However, in this case, the doctor's analysis and possible conclusions could be potentially misleading and unreliable.First, the partial data that is available on those patients could still be used. In that case, the doctor could calculate the mean, median or mode values for the missing data use them. For this doctor, the approach might be different for categorical versus continuous variables, for example. Therefore, the doctor's analysis could be bias and precaution needs to be taken whem applying the na.rm=TRUE to remove missing values.

##(LC3.3) Modify the *summarize* function to create summary_temp to also use the n() summary function: *summarize(count = n())*. What does the returned value correspond to?

```
summary_temp <- weather %>%
  summarize(mean = mean(temp, na.rm= TRUE),
            std_dev = sd(temp, na.rm = "TRUE"), count= n())
summary_temp
```

```
## # A tibble: 1 x 3
##    mean std_dev count
##   <dbl>   <dbl> <int>
## 1  55.3    17.8 26115
```

Answer The returned value of 26115 corresponds to the number or rows or observations the query pulled to calculate the mean and the standard deviation. Since there are missing values in the dataset, the query removes the missing values via "na.rm=TRUE" then uses the remaining data of 26115 observations to calculate the summary statistics.

##(LC3.4) Why doesn't the following code work? Run the code line by line instead of all at once, and then look at the data. In other words, run *summary_temp <- weather%>% summarize(mean= mean(temp, na.rm= TRUE))* first

summary_temp2 <- weather%>% summarize(mean = mean (temp, na.rm= TRUE))%>% summarize(std_dev = sd(temp, na.rm= TRUE))

Answer The issue here is that the temp which is removed when running the first portion of the code. The initial data frame "summary_temp2" that was created, contains only one column with a single corresponding value of the mean. Consequently, when we attempt to run the second part of the query to calculate the standard deviation, the system generates an error because the temp does not exist anymore in the original data frame and generates the error "is.data.frame(x): object 'temp' not found". We can then correct the script by properly summarizing the values as below:

```
summary_temp2 <- weather%>%
 summarize(Mean = mean (temp, na.rm= TRUE),
 Std_Dev = sd(temp, na.rm= TRUE))
summary_temp2
```

```
## # A tibble: 1 x 2
##    Mean Std_Dev
##   <dbl>   <dbl>
## 1  55.3    17.8
```

Group_BY ROWS

```
summary_monthly_temp <- weather %>%
  group_by(month)%>%
  summarize( Mean = mean(temp, na.rm= TRUE), Std_Dev = sd(temp, na.rm= TRUE))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
summary_monthly_temp
```

```
## # A tibble: 12 x 3
##    month  Mean Std_Dev
##    <int> <dbl>   <dbl>
##  1     1  35.6   10.2
##  2     2  34.3    6.98
##  3     3  39.9    6.25
##  4     4  51.7    8.79
##  5     5  61.8    9.68
##  6     6  72.2    7.55
##  7     7  80.1    7.12
##  8     8  74.5    5.19
##  9     9  67.4    8.47
## 10    10  60.1    8.85
## 11    11  45.0   10.4
## 12    12  38.4    9.98
```

Using the *n()* counting function.

```
by_origin <-flights%>%
  group_by(origin)%>%
  summarize(count = n())
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
by_origin
```

```
## # A tibble: 3 x 2
##   origin  count
##   <chr>   <int>
## 1 EWR    120835
## 2 JFK    111279
## 3 LGA    104662
```

Grouping by more than one variable.

```
by_origin_month <-flights%>%
  group_by(origin, month)%>%
  summarize(count = n())
```

```
## `summarise()` regrouping output by 'origin' (override with `.groups` argument)
```

```
by_origin_month
```

```
## # A tibble: 36 x 3
## # Groups:   origin [3]
##    origin month count
##    <chr>  <int> <int>
```

```
##  1 EWR        1  9893
##  2 EWR        2  9107
##  3 EWR        3 10420
##  4 EWR        4 10531
##  5 EWR        5 10592
##  6 EWR        6 10175
##  7 EWR        7 10475
##  8 EWR        8 10359
##  9 EWR        9  9550
## 10 EWR       10 10104
## # ... with 26 more rows
```

(LC3.5) Recall from Chapter 2 when we looked at plots of temperatures by months in NYC. What does the standard deviation column in the summary_monthly_temp data frame tell us about temperatures in New York City throughout the year?

```
summary_monthly_temp
```

```
## # A tibble: 12 x 3
##    month  Mean Std_Dev
##    <int> <dbl>   <dbl>
##  1     1  35.6   10.2
##  2     2  34.3    6.98
##  3     3  39.9    6.25
##  4     4  51.7    8.79
##  5     5  61.8    9.68
##  6     6  72.2    7.55
##  7     7  80.1    7.12
##  8     8  74.5    5.19
##  9     9  67.4    8.47
## 10    10  60.1    8.85
## 11    11  45.0   10.4
## 12    12  38.4    9.98
```

Answer: The standard deviation of the monthly temperature tell us how the temperatures for each month is dispersed from or centered around the mean temperature. In this case, a lower standard deviation, for examples in the months of February, March and August, indicates that the different daily temperatures for this each of these months is close to the mean monthly temperatures. On the other side, a higher standard deviation, such as in January and November indicates the daily temperatures in each of these months are spread out over a large range of values and further apart from the mean monthly temperature.

For the above question, look also at the below code

```
summary_monthly_temp2 <- weather%>%
  group_by(month)%>%
  summarize(Mean= mean(temp, na.rm = TRUE),
            Standard_Dev = sd(temp, na.rm = TRUE))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
summary_monthly_temp2
```

```
## # A tibble: 12 x 3
##    month  Mean Standard_Dev
##    <int> <dbl>        <dbl>
##  1     1  35.6        10.2
##  2     2  34.3         6.98
##  3     3  39.9         6.25
##  4     4  51.7         8.79
##  5     5  61.8         9.68
##  6     6  72.2         7.55
##  7     7  80.1         7.12
##  8     8  74.5         5.19
##  9     9  67.4         8.47
## 10    10  60.1         8.85
## 11    11  45.0        10.4
## 12    12  38.4         9.98
```

(LC3.6) What code would be required to get the mean and standard deviation temperature for each day in 2013 for NYC?

```
daily_temp <- weather %>%
  group_by(year, month, day) %>%
  summarize( Mean= mean(temp, na.rm = TRUE), Standar_Dev = sd(temp, na.rm = TRUE))
```

```
## `summarise()` regrouping output by 'year', 'month' (override with `.groups` argument)
```

```
daily_temp
```

```
## # A tibble: 364 x 5
## # Groups:   year, month [12]
##     year month   day  Mean Standar_Dev
##    <int> <int> <int> <dbl>       <dbl>
##  1  2013     1     1  37.0        4.00
##  2  2013     1     2  28.7        3.45
##  3  2013     1     3  30.0        2.58
##  4  2013     1     4  34.9        2.45
##  5  2013     1     5  37.2        4.01
##  6  2013     1     6  40.1        4.40
##  7  2013     1     7  40.6        3.68
##  8  2013     1     8  40.1        5.77
##  9  2013     1     9  43.2        5.40
## 10  2013     1    10  43.8        2.95
## # ... with 354 more rows
```

##(LC3.7) Recreate by_monthly_origin, but instead of grouping via group_by(origin, month), group variables in a different order group_by(month, origin). What differs in the resulting dataset?

```
by_monthly_orgin <-flights%>%
  group_by(month, origin)%>%
  summarize(count =n())
```

```
## `summarise()` regrouping output by 'month' (override with `.groups` argument)
```

```
by_monthly_orgin
```

```
## # A tibble: 36 x 3
## # Groups:   month [12]
##    month origin count
##    <int> <chr>  <int>
##  1     1 EWR     9893
##  2     1 JFK     9161
##  3     1 LGA     7950
##  4     2 EWR     9107
##  5     2 JFK     8421
##  6     2 LGA     7423
##  7     3 EWR    10420
##  8     3 JFK     9697
##  9     3 LGA     8717
## 10     4 EWR    10531
## # ... with 26 more rows
```

```
 by_origin_month <-flights%>%
 group_by(origin, month)%>%
 summarize(count = n())
```

```
## `summarise()` regrouping output by 'origin' (override with `.groups` argument)
```

```
by_origin_month
```

```
## # A tibble: 36 x 3
## # Groups:   origin [3]
##    origin month count
##    <chr>  <int> <int>
##  1 EWR        1  9893
##  2 EWR        2  9107
##  3 EWR        3 10420
##  4 EWR        4 10531
##  5 EWR        5 10592
##  6 EWR        6 10175
##  7 EWR        7 10475
##  8 EWR        8 10359
##  9 EWR        9  9550
## 10 EWR       10 10104
## # ... with 26 more rows
```

Answer It is interesting to see how the order in the group_by() function determines the output. When we have *group_by(origin,month)* the origin, which is this case is the first argument, takes precedent and summarizes the data frame based on the origin airport for each month. In this case, the airport of origin takes precedent.On the contrary, when we *group_by(month, origin)* the month variable takes precedent and it summarizes how flights departed from each of the airports within each month.In sum, the variable that comes first in the *group_by()* function takes priority when summarizing the data frame.

(LC3.8) How could we identify how many flights left each of the three airports for each carrier?

```
by_orgin_by_carrier <- flights%>%
  group_by(carrier, origin)%>%
  summarize(count= n())
```

```
## `summarise()` regrouping output by 'carrier' (override with `.groups` argument)
```

```
by_orgin_by_carrier
```

```
## # A tibble: 35 x 3
## # Groups:   carrier [16]
##    carrier origin count
##    <chr>   <chr>  <int>
##  1 9E      EWR     1268
##  2 9E      JFK    14651
##  3 9E      LGA     2541
##  4 AA      EWR     3487
##  5 AA      JFK    13783
##  6 AA      LGA    15459
##  7 AS      EWR      714
##  8 B6      EWR     6557
##  9 B6      JFK    42076
## 10 B6      LGA     6002
## # ... with 25 more rows
```

(LC3.9) How does the filter() operation differ from a group_by() followed by a summarize()?

Answer: The *filter()* operator narrows the number of rows in the original data frame to match the specified condition in the query. The *group_by()* followed by a *summarize()* aggregates the data frame that is of a different size and contains also different variables compared to the original data frame. Another major difference is that the *filter()* operator does not modify the original dataset as it only selects the rows. The *group_by()* operator followed by the *summarize()* outputs a report that contains new values of summaries for numerical variables.

Mutate existing variables.

```
weather <- weather%>%
  mutate(temp_in_C = (temp -32) / 1.8)
```

```
summary_monthly_temp3 <- weather%>%
  group_by(month)%>%
  summarize(Mean_tem_in_F = mean(temp, na.rm = TRUE),
            Mean_temp_in_C = mean(temp_in_C, na.rm = TRUE),
            St_dev_F = sd(temp, na.rm = TRUE),
            St_dev_C = sd(temp_in_C, na.rm = TRUE))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
summary_monthly_temp3
```

```
## # A tibble: 12 x 5
##    month Mean_tem_in_F Mean_temp_in_C St_dev_F St_dev_C
##    <int>         <dbl>          <dbl>    <dbl>    <dbl>
##  1     1          35.6           2.02    10.2      5.68
##  2     2          34.3           1.26     6.98     3.88
##  3     3          39.9           4.38     6.25     3.47
##  4     4          51.7          11.0      8.79     4.88
##  5     5          61.8          16.6      9.68     5.38
##  6     6          72.2          22.3      7.55     4.19
##  7     7          80.1          26.7      7.12     3.96
##  8     8          74.5          23.6      5.19     2.88
##  9     9          67.4          19.7      8.47     4.70
## 10    10          60.1          15.6      8.85     4.91
## 11    11          45.0           7.22    10.4      5.80
## 12    12          38.4           3.58     9.98     5.55
```

```
flights <- flights %>%
  mutate(gain= dep_delay - arr_delay)
flights
```

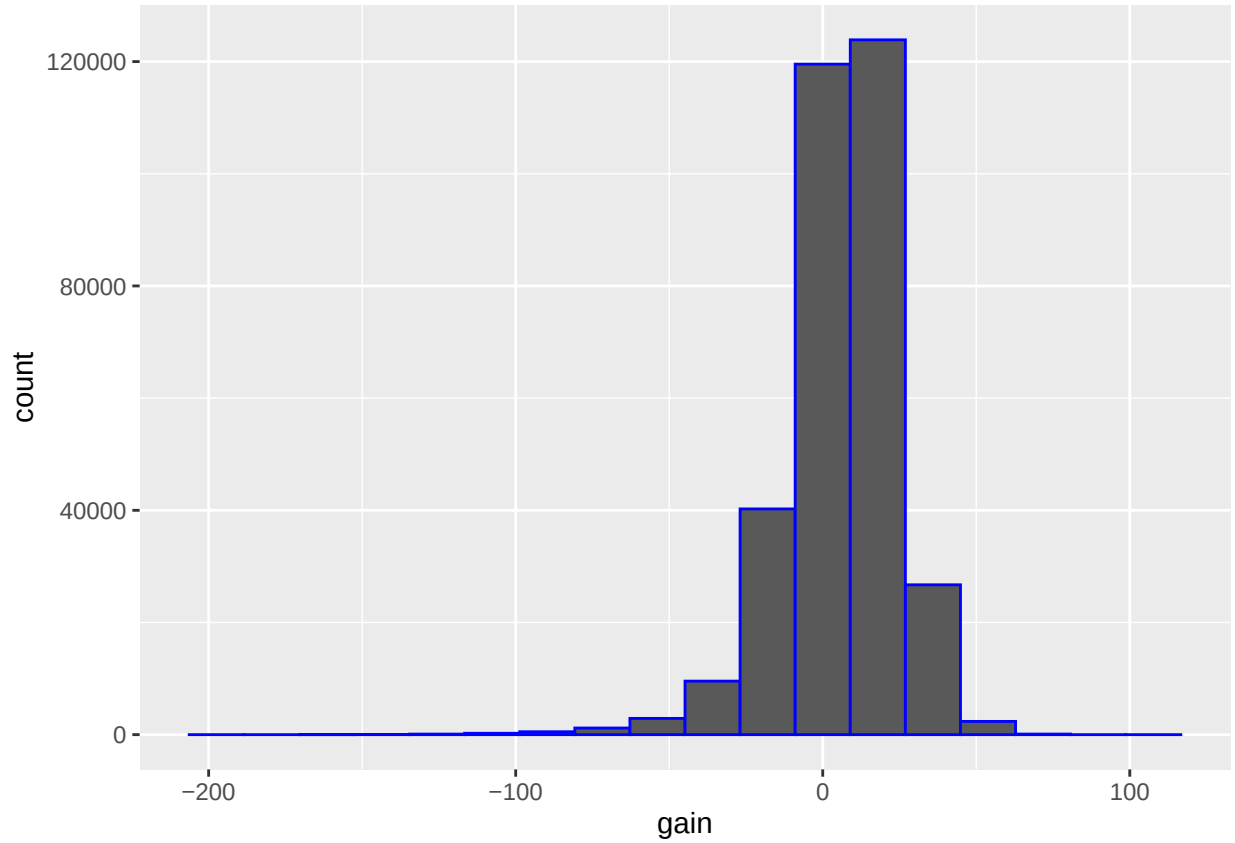
```
## # A tibble: 336,776 x 20
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1     1      517            515         2      830            819
##  2  2013     1     1      533            529         4      850            830
##  3  2013     1     1      542            540         2      923            850
##  4  2013     1     1      544            545        -1     1004           1022
##  5  2013     1     1      554            600        -6      812            837
##  6  2013     1     1      554            558        -4      740            728
##  7  2013     1     1      555            600        -5      913            854
##  8  2013     1     1      557            600        -3      709            723
##  9  2013     1     1      557            600        -3      838            846
## 10  2013     1     1      558            600        -2      753            745
## # ... with 336,766 more rows, and 12 more variables: arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>,
## #   gain <dbl>
```

```
gain_summary <- flights %>%
  summarise(min = min(gain, na.rm = TRUE),
            q1= quantile(gain, 0.25, na.rm = TRUE),
            median = quantile(gain, 0.5, na.rm = TRUE),
            q3 = quantile(gain, 0.75, na.rm = TRUE),
            max= max(gain, na.rm = TRUE),
            mean=mean(gain, na.rm = TRUE),
            sd= sd(gain, na.rm = TRUE),
            missing = sum(is.na(gain)))
gain_summary
```

```
## # A tibble: 1 x 8
##     min    q1 median    q3   max  mean    sd missing
##   <dbl> <dbl>  <dbl> <dbl> <dbl> <dbl> <dbl>   <int>
## 1  -196    -3      7    17   109  5.66  18.0    9430
```

```
ggplot(flights, aes(x=gain)) +geom_histogram(color= "blue", bins = 18)
```

```
## Warning: Removed 9430 rows containing non-finite values (stat_bin).
```




```
flights <- flights%>%
  mutate(gain= dep_delay - arr_delay,
         hours = air_time / 60,
         gain_per_hour = gain / hour
         )
```

Arrange and sort views

```
freq_dest <-flights%>%
  group_by(dest)%>%
  summarize(num_flights = n())
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
freq_dest
```

```
## # A tibble: 105 x 2
##    dest  num_flights
##    <chr>       <int>
##  1 ABQ           254
##  2 ACK           265
##  3 ALB           439
##  4 ANC             8
##  5 ATL         17215
##  6 AUS          2439
##  7 AVL           275
##  8 BDL           443
##  9 BGR           375
## 10 BHM           297
## # ... with 95 more rows
```

```
freq_dest%>%
  arrange(num_flights)
```

```
## # A tibble: 105 x 2
##    dest  num_flights
##    <chr>       <int>
##  1 LEX             1
##  2 LGA             1
##  3 ANC             8
##  4 SBN            10
##  5 HDN            15
##  6 MTJ            15
##  7 EYW            17
##  8 PSP            19
##  9 JAC            25
## 10 BZN            36
## # ... with 95 more rows
```

```
freq_dest %>%
  arrange(desc(num_flights))
```

```
## # A tibble: 105 x 2
##    dest  num_flights
##    <chr>       <int>
##  1 ORD         17283
##  2 ATL         17215
##  3 LAX         16174
##  4 BOS         15508
##  5 MCO         14082
##  6 CLT         14064
##  7 SFO         13331
##  8 FLL         12055
##  9 MIA         11728
## 10 DCA          9705
## # ... with 95 more rows
```

Join Data Frames

Inner_join

```
flights_joined <- flights %>%
  inner_join(airlines, by = "carrier")
View(flights)
View(flights_joined)
```

```
named_dests <- flights %>%
  group_by(dest)%>%
  summarize(num_flights = n())%>%
  arrange(desc(num_flights))%>%
  inner_join(airports, by = c("dest" = "faa")) %>%
  rename(airport_name = name)
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
named_dests
```

```
## # A tibble: 101 x 9
##    dest  num_flights airport_name          lat    lon   alt    tz dst   tzone
##    <chr>       <int> <chr>               <dbl>  <dbl> <dbl> <dbl> <chr> <chr>
##  1 ORD         17283 Chicago Ohare Intl   42.0  -87.9   668    -6 A     America~
##  2 ATL         17215 Hartsfield Jackson~  33.6  -84.4  1026    -5 A     America~
##  3 LAX         16174 Los Angeles Intl     33.9 -118.    126    -8 A     America~
##  4 BOS         15508 General Edward Law~  42.4  -71.0    19    -5 A     America~
##  5 MCO         14082 Orlando Intl         28.4  -81.3    96    -5 A     America~
##  6 CLT         14064 Charlotte Douglas ~  35.2  -80.9   748    -5 A     America~
##  7 SFO         13331 San Francisco Intl   37.6 -122.     13    -8 A     America~
##  8 FLL         12055 Fort Lauderdale Ho~  26.1  -80.2     9    -5 A     America~
##  9 MIA         11728 Miami Intl           25.8  -80.3     8    -5 A     America~
## 10 DCA          9705 Ronald Reagan Wash~  38.9  -77.0    15    -5 A     America~
## # ... with 91 more rows
```

```
flights_weather_joined <- flights %>%
  inner_join(weather, by = c("year", "month", "day", "hour", "origin"))
View(flights_weather_joined)
```

Thank you!

Bernardo Vimpi