Subsetting and Aggregation

Irfan Kanat

July 3, 2017

Following the first two learning activities of this module, we now know quite a bit about figuring out the characteristics of our datasets. In this learning activity we will learn about pre-processing data to obtain useful subsets and aggregations of data.

Introducing Data

You may remember the **ZRI Summary: Multifamily, SFR, Condo/Co-op (Current Month)** dataset from Assignment 1. I will use the same dataset for demonstration purposes.

```
zillow <- read.csv("data/zillow.csv")
View(zillow)
```

```
## Warning in View(zillow): X cannot set locale modifiers
```

Subsetting with Indexes

Subsetting is selecting a smaller sample from available data, be it observations (rows) or variables (columns).

Variables

We have covered filtering the data in two separate learning activities thus far. Same principles can be used to extract a subset.

We know by now various ways to extract specific columns. Here are some new twists.

Dropping a Column

Let us say I want to drop a single column (MoM).

```
colnames(zillow) # names of columns
```

```
##  [1] "Date"          "RegionName"    "State"         "Metro"
##  [5] "County"        "City"          "SizeRank"      "Zri"
##  [9] "MoM"           "QoQ"           "YoY"           "ZriRecordCnt"
```

```
# MoM is the 9th column
zillowSS0 <- zillow[, -9] # A negative indice will exclude that column
colnames(zillowSS0)
```

```
##  [1] "Date"          "RegionName"    "State"         "Metro"
##  [5] "County"        "City"          "SizeRank"      "Zri"
##  [9] "QoQ"           "YoY"           "ZriRecordCnt"
```

You can also drop multiple columns by combining negative indices.

```
zillowSS1 <- zillow[, c(-9, -10)]
colnames(zillowSS1)
```

```
##  [1] "Date"          "RegionName"    "State"         "Metro"
##  [5] "County"        "City"          "SizeRank"      "Zri"
##  [9] "YoY"           "ZriRecordCnt"
```

I will now remove these subset datasets as they are cluttering the working environment (and to demonstrate rm() function).

```
rm(zillowSS0, zillowSS1)
```

This is a bit harder to follow but I am including it here for sake reminding you about logical operators and how they work in indexes.

A way to do the same with column name.

```
# This is harder to follow, but achieves the same thing
zillowSS0 <- zillow[, !(colnames(zillow) %in% "MoM")]
colnames(zillowSS0)
```

```
##  [1] "Date"          "RegionName"    "State"         "Metro"
##  [5] "County"        "City"          "SizeRank"      "Zri"
##  [9] "QoQ"           "YoY"           "ZriRecordCnt"
```

Let us break it down.

You know what zillow[,] does, it allows us to use indexes to call various rows and columns.

Let us look at what the other parts do.

```
# This shows you if the column name matches MoM
colnames(zillow) %in% "MoM"
```

```
##  [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE
## [12] FALSE
```

```
# If you use it to index zillow you will only get MoM
# The exclamation mark reverts it, Trues become False and Falses become True
!(colnames(zillow) %in% "MoM")
```

```
##  [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE
## [12]  TRUE
```

```
# When we feed this to the index, we get all columns except "MoM"
```

You learned about two new operators %in% (matching) and ! (inverse).

Observations

You may be interested in only certain rows in a dataset.

Let us say we want to focus on rent in Ohio counties only. You remember logic operators from earlier learning activities.

```
zillowOH <- zillow[zillow$State == "OH", ]
```

zillow$State == “OH” will return an array of True/False results. It will be true where State is Ohio, and false otherwise. When we feed this logical operation into the row index, and R will only return observations where evaluation resulted in True.

We can also combine multiple criteria in our filtering. Let us say we are interested in rent in Ohio counties but only for areas where Zri is lower than $1000.

```
zillowOHCheap <- zillow[zillow$State == "OH" & zillow$Zri < 1000, ]
```

We used an & (and operator) to combine two logical operations, R only returns results where both conditions evaluate True.

Let us say you have some flexibility, you are looking for a rental in Ohio ***or*** West Virginia.

```
zillowOHWV <- zillow[zillow$State == "OH" | zillow$State == "WV", ]
```

Here we used a | (or operator) to combine two logical operations, R only returns results where at least one condition is True.

Let us clean the workspace by removing some datasets.

```
# Yeah, this may be a bit obscure at the moment. Try to figure it out, its good exercise.
rm(list = ls(pattern = "zillowOH*"))
```

Here is a little exercise for you. Considering there are 15936 observations in zillow dataset, can you make R randomly select 50 numbers between 1 and 15936 and create a random subset? Hint: read about sample function.

Subset Function

Subset function can do both variable and column subsetting at the same time and it has a more intuitive interface than indexing.

Check the manual pages.

```
?subset
```

The subset syntax is as follows:

```
subset(x, subset, select, drop)
```

- x is the data to be subsetted.
- subset is the criteria for rows.
- select is the list of columns to keep

only x is mandatory.

Selecting Variables

Let us just select State, County, and Zri from zillow dataset.

```
# Note the lack of quotation marks
zillowSS0 <- subset(zillow, select = c(State, County, Zri))
colnames(zillowSS0) # names of columns
```

```
## [1] "State"  "County" "Zri"
```

```
head(zillowSS0) # A sampling of data
```

```
##   State   County  Zri
## 1    NY New York 3535
## 2    IL     Cook 1875
## 3    NY New York 3777
## 4    IL     Cook 2106
## 5    TX  El Paso  992
## 6    NY New York 3621
```

Here I called subset with two parameters. 1. zillow: I specified which data is to be subsetted. 2. select=c(State, County, Zri): I specified which columns I wanted to keep.

One nice thing about subset is that you can use ranges of columns with names just like with ranges of numbers. Let us say we want all variables between date and zri.

```
# Note the lack of quotation marks
zillowSS1 <- subset(zillow, select = Date:Zri)
colnames(zillowSS1) # names of columns
```

```
## [1] "Date"       "RegionName" "State"      "Metro"      "County"
## [6] "City"       "SizeRank"   "Zri"
```

```
head(zillowSS1) # A sampling of data
```

```
##         Date RegionName State    Metro   County     City SizeRank  Zri
## 1 2017-05-31      10025    NY New York New York New York        0 3535
## 2 2017-05-31      60657    IL  Chicago     Cook  Chicago        1 1875
## 3 2017-05-31      10023    NY New York New York New York        2 3777
## 4 2017-05-31      60614    IL  Chicago     Cook  Chicago        3 2106
## 5 2017-05-31      79936    TX  El Paso  El Paso  El Paso        4  992
## 6 2017-05-31      10002    NY New York New York New York        5 3621
```

Here I called subset with two parameters again. 1. zillow: the dataset 2. select=Date:Zri: I specified a range of columns from Date to Zri.

```
rm(list = ls(pattern = "zillowSS*"))
```

Selecting Observations

Just as in the indexing, we can use logical operators to extract specific rows.

Let us carry out that example of Ohio or West Virginia.

```
zillowOHWV <- subset(zillow, State == "OH" | State == "WV")
head(zillowOHWV)
```

```
##           Date RegionName State     Metro   County        City SizeRank
## 80  2017-05-31      44107    OH Cleveland Cuyahoga    Lakewood       79
## 92  2017-05-31      44035    OH Cleveland   Lorain      Elyria       91
## 134 2017-05-31      43055    OH  Columbus  Licking      Newark      133
## 146 2017-05-31      44060    OH Cleveland     Lake      Mentor      145
## 147 2017-05-31      43081    OH  Columbus Franklin Westerville      146
## 163 2017-05-31      43123    OH  Columbus Franklin  Grove City      162
##      Zri          MoM          QoQ          YoY ZriRecordCnt
## 80  1211 -0.002471170 -0.004930156  0.008326395        15230
## 92   832  0.008484848  0.008484848 -0.007159905        21011
## 134  820 -0.007263923 -0.023809524 -0.060710195        20737
## 146 1336  0.001499250  0.008301887  0.028483449        24789
## 147 1485  0.007462687  0.017123288  0.029106029        20728
## 163 1334  0.004518072  0.015220700  0.024577573        21877
```

Here I called subset with two parameters 1. zillow: the dataset 2. State == “OH” | State == “WV”: State is OH or WV

Plain language translation of this command is: “get me the rows from zillow dataset where the state is either Ohio or West Virginia.”

```
rm(zillowOHWV)
```

Selecting Both

We can subset both variables and observations at the same time.

```
zillowOW <- subset(zillow,
                   (State == "OH" | State == "WV") & Zri < 1000,
                   select = Date:Zri)
```

Same idea, only made more complicated by including both row and column subsetting. Parameters: 1. zillow: dataset 2. (State==“OH” | State==“WV”) & Zri<1000: State is Ohio or West Virginia, and Zri is less than 1000 3. select=Date:Zri: columns between Date and Zri

Plain language translation of this command is: “Get me columns from Date to Zri of the zillow dataset, where the state is either Ohio or West Virginia, and the Zri is less than 1000.”

```
rm(zillowOW)
```

Aggregating

Zillow Data is at zip code level. Let us say we are interested in State level data. We do not care about specific zip codes. Should we look for state level data elsewhere or is there a way to convert the zip code level data to state level.

What we can do is to aggregate the zip code level data to obtain a State level dataset.

Aggregating allows you to come up with combined measures for broad categories from lower level observations. You can aggregate individual student scores to obtain a school level score, or company level profitability data to obtain industry level profitability measures.

When you are aggregating data you need to make certain decisions about variables. Certain variables are more meaningful as averages (average rent in OH), other variables are more meaningful as sums (population), and yet others as maximums (Dates). Some variables will need to be abandoned all together (zip codes). The correct function to use will depend on the analysis you want to do. It is a judgment call, and it is hard to teach. Just use your intuition and try to learn from your mistakes as you go.

The simplest way to aggregate data that is available in R Base is the aggregate() function. Let us inspect manual pages for aggregate().

```
?aggregate
```

aggregate() can work either with a formula or with a by parameter.

Aggregate By

We can specify the columns to aggregate by and aggregate will process all numeric columns (unless otherwise indicated).

Let us drop all non-numeric columns, you will note this is mostly lower level data such as city names that will need to be dropped anyway:

```
# Let us drop non-numeric columns
zillowSS0 <- subset(zillow, select = c(State, SizeRank:ZriRecordCnt))
```

Let us calculate average values for all other variables and save it into a new dataset.

```
# Aggregate
zillowState <- aggregate(x = zillowSS0[, -1], by = list(zillowSS0$State),
                         FUN = mean, na.rm = T)
```

Let us break the command down:

```
zillowState <- aggregate(x = zillowSS0[,-1], by = list(zillowSS0$State), FUN = mean, na.rm = T)
```

1. zillowState <- : Here I create a new dataset and assign whatever is on the other side of the assignment operator (<-) to it.
2. aggregate(): I call aggregate function with 4 parameters.

- x = zillowSS0[,-1]: Aggregate everything in zillowSS0 except for State (1st column).
- by = list(zillowSS0$State): Use State as the aggregating variable
- FUN = mean: use mean function to calcuate average values for States.
- na.rm = T: Remove missing observations (more on this later).

Translation to plain english would be: "calculate the averages of everything in zillowSS0[,-1] by State, skip missing observations in calculations."

It has 51 observations, same as the number of states... That is encouraging.

Let us see if we got it right, by calculating the average Zri for Ohio and comparing it to aggregated value of Zri.

```
mean(zillow[zillow$State == "OH", "Zri"])
```

```
## [1] 1089.942
```

```
zillowState[zillowState$Group.1 == "OH", ]
```

```
##    Group.1 SizeRank      Zri         MoM          QoQ          YoY
## 36      OH 8389.299 1089.942 0.001219884 3.043488e-05 -0.001365574
##    ZriRecordCnt
## 36     5062.327
```

Seems like we got it right!

You can change the type of calculation done on a variable by specifying a different function in FUN parameter. You can even use your own functions.

Aggregate with Formula

R uses formula interface in models, aggregate allows you to use a similar interface to aggregate data.

Formula can be thought of an equation, on the left hand side you have dependent variables and on the right, you have independent variables.

```
Zri ~ State
```

would mean estimate Zri by State.

To calculate average Zri by state, we can simply type.

```
aggregate(Zri ~ State, data = zillowSS0, FUN = mean, na.rm = T)
```

```
##    State       Zri
## 1     AK 1729.2857
## 2     AL 1027.8454
## 3     AR  964.6587
## 4     AZ 1362.9800
```

```
## 5     CA 2525.3384
## 6     CO 1767.6952
## 7     CT 1899.4776
## 8     DC 2764.6190
## 9     DE 1433.6304
## 10    FL 1568.0983
## 11    GA 1235.6296
## 12    HI 2243.1351
## 13    IA 1130.0273
## 14    ID 1138.3100
## 15    IL 1437.0751
## 16    IN 1037.0146
## 17    KS 1114.6590
## 18    KY 1087.7676
## 19    LA 1229.5794
## 20    MA 2145.7942
## 21    MD 1789.1417
## 22    ME 1456.4200
## 23    MI 1144.9263
## 24    MN 1416.2854
## 25    MO 1095.7138
## 26    MS 1073.4712
## 27    MT 1234.8889
## 28    NC 1132.7871
## 29    ND 1321.1000
## 30    NE 1258.2737
## 31    NH 1604.3387
## 32    NJ 2193.2574
## 33    NM 1200.0123
## 34    NV 1350.3551
## 35    NY 1962.0446
## 36    OH 1089.9417
## 37    OK  959.1245
## 38    OR 1555.8839
## 39    PA 1271.4369
## 40    RI 1698.0357
## 41    SC 1198.5627
## 42    SD 1220.8333
## 43    TN 1169.5714
## 44    TX 1437.7267
## 45    UT 1484.7500
## 46    VA 1504.2735
## 47    VT 1572.1071
## 48    WA 1695.9591
## 49    WI 1228.4759
## 50    WV  994.7835
## 51    WY 1116.1304
```

To obtain the same result as the example in Aggregate By section we can use a . (dot) notation. Rather than giving a list of variables we can place a . to signify all variables in data.

```
aggregate(.~State, data = zillowSS0, FUN = mean, na.rm = T)
```

```
##    State  SizeRank       Zri           MoM           QoQ           YoY
## 1     AK  6593.429 1729.2857 -1.243602e-03 -4.569654e-03 -0.0514966006
```

```
## 2     AL  8666.055 1027.8454  2.327414e-03  5.190656e-03 -0.0072165342
## 3     AR  9068.102  964.6587 -1.896631e-03 -2.743182e-03 -0.0195052515
## 4     AZ  5520.580 1362.9800  1.723384e-03  3.657818e-03  0.0210472573
## 5     CA  5468.302 2525.3384  3.767425e-03  1.058260e-02  0.0356144772
## 6     CO  7498.643 1767.6952  2.241286e-03  5.101782e-03  0.0102028682
## 7     CT  8487.245 1899.4776 -3.315680e-05  4.205537e-04 -0.0127052616
## 8     DC  3736.476 2764.6190  3.220500e-03  7.193089e-03  0.0248985950
## 9     DE  6602.783 1433.6304  2.889184e-03  6.866306e-03 -0.0284940667
## 10    FL  5356.305 1568.0983  7.732646e-04  5.042015e-03  0.0179015613
## 11    GA  7181.185 1235.6296  2.187278e-03  7.099238e-03  0.0168325483
## 12    HI  6125.838 2243.1351  2.211295e-03  9.604554e-03  0.0126819498
## 13    IA 10890.027 1130.0273 -5.267372e-04  1.995419e-03 -0.0269299656
## 14    ID  9255.240 1138.3100  3.975496e-03  1.119792e-02  0.0395984708
## 15    IL  8452.895 1437.0751 -2.169168e-04 -1.274634e-04 -0.0264761835
## 16    IN  9183.721 1037.0146  2.078078e-03  4.203470e-03  0.0022105917
## 17    KS  9992.051 1114.6590 -5.896126e-04  4.053500e-03 -0.0151921056
## 18    KY  7847.600 1087.7676  1.825053e-03  1.368610e-03 -0.0002676517
## 19    LA  7639.117 1229.5794  4.542072e-06 -5.138813e-03 -0.0439533545
## 20    MA  8101.535 2145.7942  2.030645e-03  6.359866e-03  0.0159528372
## 21    MD  8865.226 1789.1417 -8.698668e-05  1.728685e-04 -0.0028587447
## 22    ME 11003.270 1456.4200  8.150521e-03  2.032671e-02  0.0500954910
## 23    MI  8855.704 1144.9263  1.679335e-03  6.996579e-03  0.0129405413
## 24    MN  9969.530 1416.2854  1.840533e-03  9.812834e-03  0.0019923182
## 25    MO  7966.014 1095.7138  1.447715e-03  4.597780e-03 -0.0027756457
## 26    MS  7577.904 1073.4712  6.028350e-05  1.376988e-03 -0.0276932537
## 27    MT  8769.130 1234.8889  3.320638e-03  8.806315e-03  0.0275279026
## 28    NC  7775.232 1132.7871  1.834659e-03  3.990841e-03 -0.0051514793
## 29    ND  5474.400 1321.1000  4.845916e-03  7.377523e-03 -0.0103710039
## 30    NE  8701.126 1258.2737  1.928938e-03  2.150525e-03  0.0026188898
## 31    NH 11594.532 1604.3387  2.041624e-03  1.053728e-02  0.0191615401
## 32    NJ  7923.279 2193.2574 -4.116427e-04 -1.376008e-03 -0.0008977488
## 33    NM  7026.272 1200.0123 -2.122567e-04  4.810052e-05 -0.0295133756
## 34    NV  5638.879 1350.3551  5.278858e-03  1.358766e-02  0.0254922100
## 35    NY  8715.132 1962.0446 -8.092522e-04  9.785469e-04 -0.0048725708
## 36    OH  8389.299 1089.9417  1.219884e-03  3.043488e-05 -0.0013655739
## 37    OK  9342.782  959.1245 -2.582289e-03 -9.205668e-03 -0.0458668979
## 38    OR  7839.429 1555.8839  4.172058e-03  1.218142e-02  0.0444798268
## 39    PA  9118.199 1271.4369 -1.661727e-03 -4.401752e-03 -0.0211231537
## 40    RI  6901.661 1698.0357  4.124864e-03  1.037611e-02  0.0257775743
## 41    SC  7835.639 1198.5627  6.004260e-03  1.436448e-02  0.0086613559
## 42    SD  8255.083 1220.8333  2.462333e-03  1.041313e-02  0.0264755090
## 43    TN  8301.026 1169.5714  2.342292e-03  6.646830e-03  0.0046478446
## 44    TX  6427.314 1437.7267 -9.621785e-05 -3.126355e-05 -0.0091652268
## 45    UT  6402.212 1484.7500  5.815598e-03  1.411483e-02  0.0439552950
## 46    VA  8680.649 1504.2735  2.979883e-04  1.340462e-03 -0.0095722977
## 47    VT 12668.464 1572.1071  2.510030e-02  5.437123e-02  0.0328119972
## 48    WA  7237.112 1695.9591  5.266344e-03  1.484918e-02  0.0507619324
## 49    WI  9671.043 1228.4759 -1.334457e-03  6.814510e-03  0.0184755370
## 50    WV  9214.948  994.7835 -3.433177e-04  4.059388e-03 -0.0250154636
## 51    WY  8017.304 1116.1304 -3.613409e-03 -7.243096e-03 -0.0937681226
##    ZriRecordCnt
## 1      5863.810
## 2      4417.931
## 3      3949.904
```

```
## 4      8140.864
## 5      7422.763
## 6      6119.026
## 7      4738.992
## 8      8483.000
## 9      6460.478
## 10     8874.542
## 11     6346.508
## 12     8368.324
## 13     3335.877
## 14     4627.090
## 15     5846.171
## 16     5008.080
## 17     3597.599
## 18     5469.935
## 19     4702.360
## 20     4852.966
## 21     5749.251
## 22     2957.830
## 23     5330.170
## 24     3918.050
## 25     5471.078
## 26     4486.750
## 27     3754.130
## 28     5667.008
## 29     5126.900
## 30     5725.368
## 31     2806.581
## 32     5233.002
## 33     6502.481
## 34     8247.720
## 35     4368.514
## 36     5062.327
## 37     4198.296
## 38     4951.723
## 39     4305.296
## 40     5698.286
## 41     5693.612
## 42     4039.917
## 43     5918.894
## 44     6354.086
## 45     7373.404
## 46     5101.620
## 47     1491.488
## 48     5946.793
## 49     3551.791
## 50     3529.928
## 51     4644.478
```

Let us go over the function call:

```
aggregate(.~State, data=zillowSS0, FUN= mean, na.rm=T)
```

1. aggregate(): I call the aggregate function with 4 parameters.

- .~State:

  - . : means all variables
  - ~ : is like an equals sign in a formula, it translates to estimate whats to my left by whats to my right
  - State : What will be used to group by, you can add more variables with a + sign.
- data = zillowSS0: use zillowSS0 dataset to find variables
- FUN = mean: use mean function to aggregate, alternatives can be any function such as sum, max, min...
- na.rm = TRUE: Skip missing observations.

Personally, I find the formula interface to be cleaner.

Further Reading

If you want to see more examples, refer to Data Manipulation (terrible name choice) section of my R Workshop. The material was aimed at a more advanced audience, but with the foundations laid in this section you can follow along with it.

Solutions to Exercises

1-Can you make R randomly select 50 numbers between 1 and 15936 and create a random subset?

```
# Setting seed for random number generator to ensure reproducability
set.seed(2017)
zillowRandom <- zillow[sample(1:nrow(zillow), 50), ]
```