Window Functions

Data Wrangling and Husbandry

2/17/2020

- Window functions are a concept borrowed from SQL
- Not to be confused with sliding windows used in smoothing, for example

A classification of functions on vectors

- aggregation functions
  - e.g., `sum()`, `mean()`, `n()`, `max()`
  - take n inputs and return a single value
- functions that work element-wise
  - e.g., `+`, `exp()`, `round()`
  - takes n inputs and returns n outputs
- window functions
  - Unlike aggregation functions, window functions return n values
  - Unlike element-wise functions, the output depends on all of the input values

```r
# Example from the vignette

library(Lahman)
batting <- Batting %>% as_tibble %>%
  select(playerID, yearID, teamID, G, AB:H)
batting <- batting %>% arrange(playerID, yearID, teamID)
batting[1:3, ]
```

```
## # A tibble: 3 x 7
##   playerID  yearID teamID     G    AB     R     H
##   <chr>      <int> <fct>  <int> <int> <int> <int>
## 1 aardsda01   2004 SFN       11     0     0     0
## 2 aardsda01   2006 CHN       45     2     0     0
## 3 aardsda01   2007 CHA       25     0     0     0
```

```r
players <- batting %>% group_by(playerID)
```

```
# Within each player, rank each year by the number of game
players %>% mutate(G_rank = min_rank(G))
```

```
## # A tibble: 105,861 x 8
## # Groups:   playerID [19,428]
##    playerID  yearID teamID     G    AB     R     H G_ran
##    <chr>      <int> <fct>  <int> <int> <int> <int>  <int
##  1 aardsda01   2004 SFN       11     0     0     0
##  2 aardsda01   2006 CHN       45     2     0     0
##  3 aardsda01   2007 CHA       25     0     0     0
##  4 aardsda01   2008 BOS       47     1     0     0
##  5 aardsda01   2009 SEA       73     0     0     0
##  6 aardsda01   2010 SEA       53     0     0     0
##  7 aardsda01   2012 NYA        1     0     0     0
##  8 aardsda01   2013 NYN       43     0     0     0
##  9 aardsda01   2015 ATL       33     1     0     0
## 10 aaronha01   1954 ML1      122   468    58   131
## # ... with 105,851 more rows
```

```r
# Within each player, rank each year by the number of games
players %>% mutate(G_rank = min_rank(desc(G))) %>%
  arrange(playerID, G_rank)
```

```
## # A tibble: 105,861 x 8
## # Groups:   playerID [19,428]
##    playerID  yearID teamID     G    AB     R     H G_ran
##    <chr>      <int> <fct>  <int> <int> <int> <int>  <int
##  1 aardsda01   2009 SEA       73     0     0     0
##  2 aardsda01   2010 SEA       53     0     0     0
##  3 aardsda01   2008 BOS       47     1     0     0
##  4 aardsda01   2006 CHN       45     2     0     0
##  5 aardsda01   2013 NYN       43     0     0     0
##  6 aardsda01   2015 ATL       33     1     0     0
##  7 aardsda01   2007 CHA       25     0     0     0
##  8 aardsda01   2004 SFN       11     0     0     0
##  9 aardsda01   2012 NYA        1     0     0     0
## 10 aaronha01   1963 ML1      161   631   121   201
## # ... with 105,851 more rows
```

```r
# For each player, find the two years with most hits
players %>% filter(min_rank(desc(H)) <= 2 & H > 0)
```

```
## # A tibble: 26,466 x 7
## # Groups:   playerID [14,771]
##    playerID  yearID teamID     G    AB     R     H
##    <chr>      <int> <fct>  <int> <int> <int> <int>
##  1 aaronha01   1959 ML1      154   629   116   223
##  2 aaronha01   1963 ML1      161   631   121   201
##  3 aaronto01   1962 ML1      141   334    54    77
##  4 aaronto01   1968 ATL       98   283    21    69
##  5 abadan01    2003 BOS        9    17     1     2
##  6 abadfe01    2012 HOU       37     7     0     1
##  7 abadijo01   1875 BR2        1     4     1     1
##  8 abadijo01   1875 PH3       11    45     3    10
##  9 abbated01   1904 BSN      154   579    76   148
## 10 abbated01   1905 BSN      153   610    70   170
## # ... with 26,456 more rows
```

```r
# For each player, find every year with more games than th
players %>% filter( G > lag(G))
```

```
## # A tibble: 42,031 x 7
## # Groups:   playerID [12,394]
##    playerID  yearID teamID     G    AB     R     H
##    <chr>      <int> <fct>  <int> <int> <int> <int>
##  1 aardsda01   2006 CHN       45     2     0     0
##  2 aardsda01   2008 BOS       47     1     0     0
##  3 aardsda01   2009 SEA       73     0     0     0
##  4 aardsda01   2013 NYN       43     0     0     0
##  5 aaronha01   1955 ML1      153   602   105   189
##  6 aaronha01   1958 ML1      153   601   109   196
##  7 aaronha01   1959 ML1      154   629   116   223
##  8 aaronha01   1961 ML1      155   603   115   197
##  9 aaronha01   1962 ML1      156   592   127   191
## 10 aaronha01   1963 ML1      161   631   121   201
## # ... with 42,021 more rows
```

```r
# For each player, compute avg change in games played per
players %>% mutate(G_change = (G - lag(G)) / (yearID - lag(
```

```
## # A tibble: 105,861 x 8
## # Groups:   playerID [19,428]
##    playerID  yearID teamID     G    AB     R     H G_cha
##    <chr>      <int> <fct>  <int> <int> <int> <int>    <c
##  1 aardsda01   2004 SFN       11     0     0     0
##  2 aardsda01   2006 CHN       45     2     0     0
##  3 aardsda01   2007 CHA       25     0     0     0
##  4 aardsda01   2008 BOS       47     1     0     0
##  5 aardsda01   2009 SEA       73     0     0     0
##  6 aardsda01   2010 SEA       53     0     0     0
##  7 aardsda01   2012 NYA        1     0     0     0
##  8 aardsda01   2013 NYN       43     0     0     0
##  9 aardsda01   2015 ATL       33     1     0     0
## 10 aaronha01   1954 ML1      122   468    58   131
## # ... with 105,851 more rows
```

```r
# For each player, find all where they played more games th
# (doesn't actually use a window function)
players %>% filter(G > mean(G))
```

```
## # A tibble: 52,228 x 7
## # Groups:   playerID [14,192]
##    playerID  yearID teamID     G    AB     R     H
##    <chr>      <int> <fct>  <int> <int> <int> <int>
##  1 aardsda01   2006 CHN       45     2     0     0
##  2 aardsda01   2008 BOS       47     1     0     0
##  3 aardsda01   2009 SEA       73     0     0     0
##  4 aardsda01   2010 SEA       53     0     0     0
##  5 aardsda01   2013 NYN       43     0     0     0
##  6 aaronha01   1955 ML1      153   602   105   189
##  7 aaronha01   1956 ML1      153   609   106   200
##  8 aaronha01   1957 ML1      151   615   118   198
##  9 aaronha01   1958 ML1      153   601   109   196
## 10 aaronha01   1959 ML1      154   629   116   223
## # ... with 52,218 more rows
```

```r
# For each, player compute a z score based on number of ga
# (doesn't actually use a window function)
mutate(players, G_z = (G - mean(G)) / sd(G))
```

```
## # A tibble: 105,861 x 8
## # Groups:   playerID [19,428]
##    playerID  yearID teamID     G    AB     R     H    G_
##    <chr>      <int> <fct>  <int> <int> <int> <int>  <db
##  1 aardsda01   2004 SFN       11     0     0     0 -1.17
##  2 aardsda01   2006 CHN       45     2     0     0  0.37
##  3 aardsda01   2007 CHA       25     0     0     0 -0.53
##  4 aardsda01   2008 BOS       47     1     0     0  0.46
##  5 aardsda01   2009 SEA       73     0     0     0  1.64
##  6 aardsda01   2010 SEA       53     0     0     0  0.73
##  7 aardsda01   2012 NYA        1     0     0     0 -1.62
##  8 aardsda01   2013 NYN       43     0     0     0  0.28
##  9 aardsda01   2015 ATL       33     1     0     0 -0.17
## 10 aaronha01   1954 ML1      122   468    58   131 -1.16
## # ... with 105,851 more rows
```

The most useful window functions 1/2

- `lead()` and `lag()`

```r
letters[1:10]
```

```
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j"
```

```r
lead(letters[1:10])
```

```
##  [1] "b" "c" "d" "e" "f" "g" "h" "i" "j" NA
```

```r
lead(letters[1:10], n = 2)
```

```
##  [1] "c" "d" "e" "f" "g" "h" "i" "j" NA  NA
```

```r
lag(letters[1:10])
```

```
##  [1] NA  "a" "b" "c" "d" "e" "f" "g" "h" "i"
```

```r
# Compute the relative change in games played
players %>% mutate(G_delta = G - lag(G))
```

```
## # A tibble: 105,861 x 8
## # Groups:   playerID [19,428]
##    playerID  yearID teamID     G    AB     R     H G_del
##    <chr>      <int> <fct>  <int> <int> <int> <int>   <i
##  1 aardsda01   2004 SFN       11     0     0     0
##  2 aardsda01   2006 CHN       45     2     0     0
##  3 aardsda01   2007 CHA       25     0     0     0     -
##  4 aardsda01   2008 BOS       47     1     0     0
##  5 aardsda01   2009 SEA       73     0     0     0
##  6 aardsda01   2010 SEA       53     0     0     0     -
##  7 aardsda01   2012 NYA        1     0     0     0     -
##  8 aardsda01   2013 NYN       43     0     0     0
##  9 aardsda01   2015 ATL       33     1     0     0     -
## 10 aaronha01   1954 ML1      122   468    58   131
## # ... with 105,851 more rows
```

```r
# Find when a player changed teams
players %>% filter(teamID != lag(teamID))
```

```
## # A tibble: 32,475 x 7
## # Groups:   playerID [10,877]
##    playerID  yearID teamID     G    AB     R     H
##    <chr>      <int> <fct>  <int> <int> <int> <int>
##  1 aardsda01   2006 CHN       45     2     0     0
##  2 aardsda01   2007 CHA       25     0     0     0
##  3 aardsda01   2008 BOS       47     1     0     0
##  4 aardsda01   2009 SEA       73     0     0     0
##  5 aardsda01   2012 NYA        1     0     0     0
##  6 aardsda01   2013 NYN       43     0     0     0
##  7 aardsda01   2015 ATL       33     1     0     0
##  8 aaronha01   1966 ATL      158   603   117   168
##  9 aaronha01   1975 ML4      137   465    45   109
## 10 aaronto01   1968 ATL       98   283    21    69
## # ... with 32,465 more rows
```

The most useful window functions 2/2

- Ranking functions

Base R:

```
rank(x, na.last = TRUE,
     ties.method = c("average", "first", "last", "random",
```

```
x <- c(1, 1, 2, 2, 2)
rank(x, ties.method = "average")
```

```
## [1] 1.5 1.5 4.0 4.0 4.0
```

```
rank(x, ties.method = "first")  # first occurrence wins
```

```
## [1] 1 2 3 4 5
```

```r
x
```

```
## [1] 1 1 2 2 2
```

```r
rank(x, ties.method = "last") # last occurrence wins
```

```
## [1] 2 1 5 4 3
```

```r
rank(x, ties.method = "max")
```

```
## [1] 2 2 5 5 5
```

```r
rank(x, ties.method = "min") # as in sports
```

```
## [1] 1 1 3 3 3
```

The `dplyr` package has versions of most of these (not average rank, though), designed to require less typing and to align with the names of SQL functions.

```r
row_number(x)
```

```
## [1] 1 2 3 4 5
```

```r
min_rank(x)
```

```
## [1] 1 1 3 3 3
```

```r
dense_rank(x)
```

```
## [1] 1 1 2 2 2
```

- `percent_rank()` rescales `min_rank` to [0, 1]
- `cume_dist()` gives the proportion of values less than or equal to the current value.

```r
x
```

```
## [1] 1 1 2 2 2
```

```r
percent_rank(x)
```

```
## [1] 0.0 0.0 0.5 0.5 0.5
```

```r
cume_dist(x)
```

```
## [1] 0.4 0.4 1.0 1.0 1.0
```

```r
# Selects best two years
players %>% filter(min_rank(desc(G)) <= 2)
```

```
## # A tibble: 34,858 x 7
## # Groups:   playerID [19,428]
##    playerID  yearID teamID     G    AB     R     H
##    <chr>      <int> <fct>  <int> <int> <int> <int>
##  1 aardsda01   2009 SEA       73     0     0     0
##  2 aardsda01   2010 SEA       53     0     0     0
##  3 aaronha01   1963 ML1      161   631   121   201
##  4 aaronha01   1968 ATL      160   606    84   174
##  5 aaronto01   1962 ML1      141   334    54    77
##  6 aaronto01   1968 ATL       98   283    21    69
##  7 aasedo01    1985 BAL       54     0     0     0
##  8 aasedo01    1986 BAL       66     0     0     0
##  9 abadan01    2003 BOS        9    17     1     2
## 10 abadan01    2006 CIN        5     3     0     0
## # ... with 34,848 more rows
```

```r
# Selects best 10% of years
players %>% filter(cume_dist(desc(G)) <= 0.1)
```

```
## # A tibble: 3,868 x 7
## # Groups:   playerID [3,634]
##    playerID  yearID teamID     G    AB     R     H
##    <chr>      <int> <fct>  <int> <int> <int> <int>
##  1 aaronha01   1963 ML1      161   631   121   201
##  2 aaronha01   1968 ATL      160   606    84   174
##  3 aasedo01    1986 BAL       66     0     0     0
##  4 abbated01   1904 BSN      154   579    76   148
##  5 abbotgl01   1977 SEA       36     0     0     0
##  6 abbotji01   1991 CAL       34     0     0     0
##  7 abbotku01   1995 FLO      120   420    60   107
##  8 abbotpa01   2000 SEA       35     5     1     2
##  9 abernte02   1965 CHN       84    18     1     3
## 10 abramca01   1953 PIT      119   448    66   128
## # ... with 3,858 more rows
```

Finally, `ntile()` divides the vector into n buckets

```r
ntile(1:100, n = 4)
```

```
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [38] 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 3 3
##  [75] 3 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
```

This can be useful to summarize data by quantiles . . .

```r
Batting %>%
  filter(yearID >= 1961 & AB > 0) %>%
  group_by(quartile = ntile(G, 4)) %>%
  summarise(batting_average = mean(H/AB))
```

```
## # A tibble: 4 x 2
##   quartile batting_average
##      <int>           <dbl>
## 1        1           0.160
## 2        2           0.166
## 3        3           0.215
## 4        4           0.268
```

In class exercise

- Make a tibble out of the NYC restaurant health inspection dataset dropping the varibles `VIOLATION CODE`, `VIOLATION DESCRIPTION`, and `CRITICAL FLAG` and then apply the `distinct()` function to get distinct row.
- Using this new tibble, form a new tibble with the two most recent inspections for each restaurant.
- Dividing these restaurant/inspections into quintiles (meaning 5 bins) of score, find the mean score for each quintile.
- Now use only the most recent inspection and divide into quintiles per boro.