Programming in R Part I

Justin Post

August 9, 2017

Course Schedule

Day's agenda:

- 10-11:10 Session
- 10-minute break
- 11:20-12:30 Session
- 12:30-1:45 Lunch
- 1:45-2:55 Session
- 10-minute break
- 3:05-4:15 Session

What do we want to be able to do?

- Restructure Data/Clean Data
- Streamline repeated sections of code
- Improve efficiency of code
- Write custom functions to simplify code

Where do we start?

- Review of concepts
- Using dplyr/tidry to manipulate data
- For loops
- If/Then logic
- Vectorized Functions
- Function Writing
- Parallel Computing

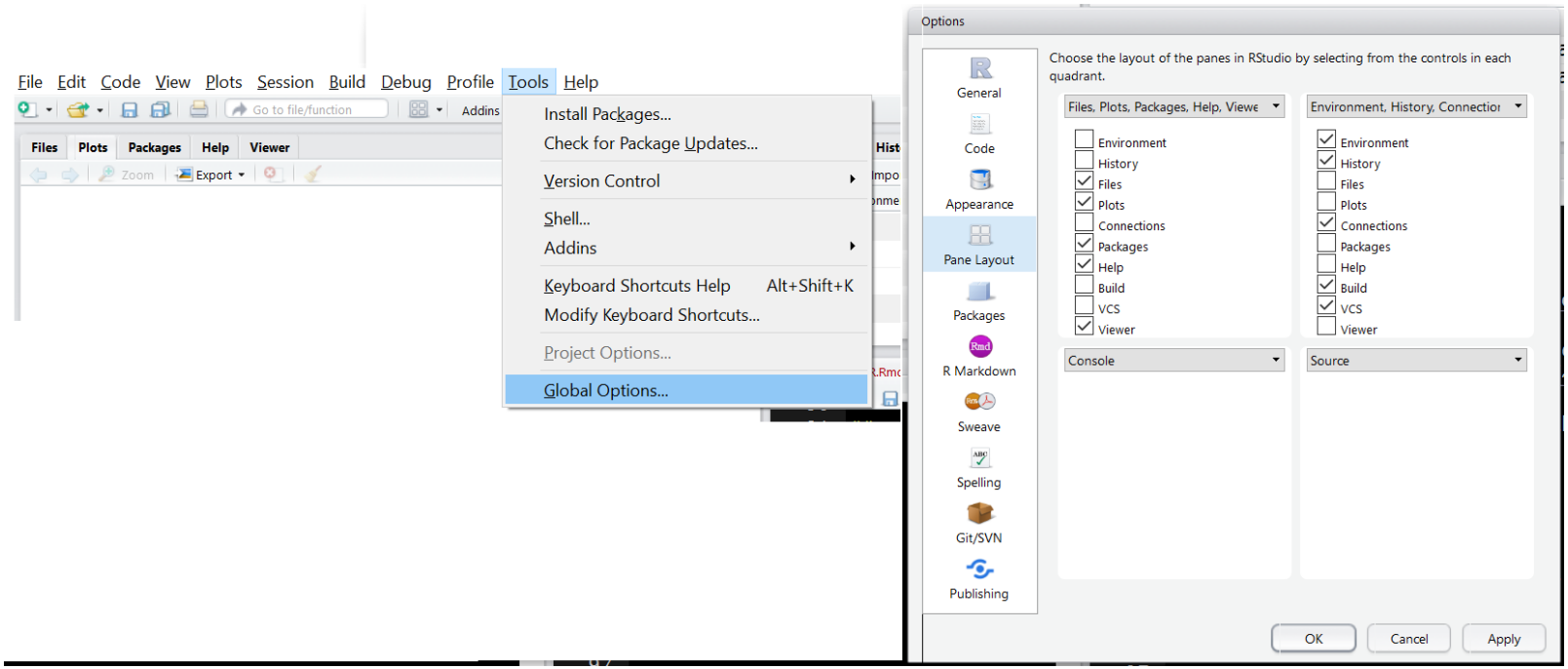
Review of Concepts

R Studio

- Great integrated development environment (IDE)
- Four main 'areas' we'll use
    - Scripting and Viewing Area
    - Workspace/History
    - Plots/Help
    - Console

Review of Concepts

R Studio - Can rearrange panes



- Global options –> Appearance allows font/background changes

Review of Concepts

Data Frames

- Best R object for data sets
- Collection (list) of vectors of the same **length**

```
x <- c("a", "b", "c", "d", "e", "f")
y <- c(1, 3, 4, -1, 5, 6)
z <- 10:15
data.frame(char = x, data1 = y, data2 = z)
```

```
##   char data1 data2
## 1    a     1    10
## 2    b     3    11
## 3    c     4    12
## 4    d    -1    13
## 5    e     5    14
## 6    f     6    15
```

Review of Concepts

Data Frames

- Consider the built in `iris` data set

```
iris
```

```
##     Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
## 1            5.1         3.5          1.4         0.2     setosa
## 2            4.9         3.0          1.4         0.2     setosa
## 3            4.7         3.2          1.3         0.2     setosa
## 4            4.6         3.1          1.5         0.2     setosa
## 5            5.0         3.6          1.4         0.2     setosa
## 6            5.4         3.9          1.7         0.4     setosa
## 7            4.6         3.4          1.4         0.3     setosa
## 8            5.0         3.4          1.5         0.2     setosa
## 9            4.4         2.9          1.4         0.2     setosa
## 10           4.9         3.1          1.5         0.1     setosa
## 11           5.4         3.7          1.5         0.2     setosa
## 12           4.8         3.4          1.6         0.2     setosa
## 13           4.8         3.0          1.4         0.1     setosa
```

Review of Concepts

Data Frames

- Can see info about object with `str()` and `attributes()`

```
str(iris)
```

```
## 'data.frame':    150 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

```
attributes(iris)
```

```
## $names
## [1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"
## [5] "Species"
##
## $row.names
##   [1]   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
##  [18]  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34
##  [35]  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51
##  [52]  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68
##  [69]  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85
##  [86]  86  87  88  89  90  91  92  93  94  95  96  97  98  99 100 101 102
## [103] 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119
## [120] 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136
## [137] 137 138 139 140 141 142 143 144 145 146 147 148 149 150
##
## $class
## [1] "data.frame"
```

Review of Concepts

Data Frames

- Accessing elements: multiple ways

```
iris[1:4, 2:4]
```

```
##   Sepal.Width Petal.Length Petal.Width
## 1         3.5          1.4         0.2
## 2         3.0          1.4         0.2
## 3         3.2          1.3         0.2
## 4         3.1          1.5         0.2
```

Review of Concepts

Data Frames

- Accessing elements: multiple ways

```
iris[1, ]
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
```

Review of Concepts

Data Frames

- Accessing elements: multiple ways

```
iris[ , c("Sepal.Length", "Species")]
```

```
##     Sepal.Length    Species
## 1            5.1     setosa
## 2            4.9     setosa
## 3            4.7     setosa
## 4            4.6     setosa
## 5            5.0     setosa
## 6            5.4     setosa
## 7            4.6     setosa
## 8            5.0     setosa
## 9            4.4     setosa
## 10           4.9     setosa
## 11           5.4     setosa
## 12           4.8     setosa
## 13           4.8     setosa
```

Review of Concepts

Data Frames

- Accessing elements: multiple ways

```
iris$Sepal.Length
```

```
##   [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9 5.4 4.8 4.8 4.3 5.8 5.7 5.4
##  [18] 5.1 5.7 5.1 5.4 5.1 4.6 5.1 4.8 5.0 5.0 5.2 5.2 4.7 4.8 5.4 5.2 5.5
##  [35] 4.9 5.0 5.5 4.9 4.4 5.1 5.0 4.5 4.4 5.0 5.1 4.8 5.1 4.6 5.3 5.0 7.0
##  [52] 6.4 6.9 5.5 6.5 5.7 6.3 4.9 6.6 5.2 5.0 5.9 6.0 6.1 5.6 6.7 5.6 5.8
##  [69] 6.2 5.6 5.9 6.1 6.3 6.1 6.4 6.6 6.8 6.7 6.0 5.7 5.5 5.5 5.8 6.0 5.4
##  [86] 6.0 6.7 6.3 5.6 5.5 5.5 6.1 5.8 5.0 5.6 5.7 5.7 6.2 5.1 5.7 6.3 5.8
## [103] 7.1 6.3 6.5 7.6 4.9 7.3 6.7 7.2 6.5 6.4 6.8 5.7 5.8 6.4 6.5 7.7 7.7
## [120] 6.0 6.9 5.6 7.7 6.3 6.7 7.2 6.2 6.1 6.4 7.2 7.4 7.9 6.4 6.3 6.1 7.7
## [137] 6.3 6.4 6.0 6.9 6.7 6.9 5.8 6.8 6.7 6.7 6.3 6.5 6.2 5.9
```

Review of Concepts

Packages - Many ways to accomplish the same thing in R

- How to choose?
  - Want 'fast' code
  - Want 'easy' syntax
  - Good default settings on functions
- Base R has reasonable defaults and syntax but functions are slow
- "TidyVerse" - collection of R packages that share common philosophies and are designed to work together!
  - Very efficient code
  - Common syntax

Review of Concepts

- If not installed (downloaded) on computer

```
install.packages("tidyverse")
```

Review of Concepts

- Once installed, `library()` or `require()` to load

```
library(tidyverse)
```

```
## Loading tidyverse: ggplot2
## Loading tidyverse: tibble
## Loading tidyverse: tidyr
## Loading tidyverse: readr
## Loading tidyverse: purrr
## Loading tidyverse: dplyr
```

```
## Conflicts with tidy packages ----------------------------------------------
```

```
## filter(): dplyr, stats
## lag():    dplyr, stats
```

Tidyverse Syntax

- Reason to prefer `dplyr` and packages from the `tidyverse`
- Fast!
- Good defaults
- All packages have similar syntax! All work on `tibbles` (data frames)
- Syntax: function(data.frame, actions, …)

Subsetting/Manipulating Data

`tbl_df()` - convert data frame to one with better printing

- If data read in with haven, readxl, or readr already in this format!
- Just 'wrap' data frame

```
#install.packages("Lahman")
library(Lahman)
head(Batting, n = 4) #look at just first 4 observations
```

Subsetting/Manipulating Data

`tbl_df()` - convert data frame to one with better printing

```
head(Batting, n = 4) #look at just first 4 observations
```

```
##    playerID yearID stint teamID lgID  G  AB  R  H X2B X3B HR RBI SB CS BB
## 1 abercda01   1871     1    TRO   NA  1   4  0  0   0   0  0   0  0  0  0
## 2  addybo01   1871     1    RC1   NA 25 118 30 32   6   0  0  13  8  1  4
## 3 allisar01   1871     1    CL1   NA 29 137 28 40   4   5  0  19  3  1  2
## 4 allisdo01   1871     1    WS3   NA 27 133 28 44  10   2  2  27  1  1  0
##   SO IBB HBP SH SF GIDP
## 1  0  NA  NA NA NA   NA
## 2  0  NA  NA NA NA   NA
## 3  5  NA  NA NA NA   NA
## 4  2  NA  NA NA NA   NA
```

Subsetting/Manipulating Data

```
Batting <- tbl_df(Batting)
Batting
```

```
## # A tibble: 101,332 × 22
##    playerID yearID stint teamID   lgID     G    AB     R     H   X2B   X3B
##       <chr>  <int> <int> <fctr> <fctr> <int> <int> <int> <int> <int> <int>
## 1 abercda01   1871     1    TRO     NA     1     4     0     0     0     0
## 2  addybo01   1871     1    RC1     NA    25   118    30    32     6     0
## 3 allisar01   1871     1    CL1     NA    29   137    28    40     4     5
## 4 allisdo01   1871     1    WS3     NA    27   133    28    44    10     2
## 5 ansonca01   1871     1    RC1     NA    25   120    29    39    11     3
## # ... with 1.013e+05 more rows, and 11 more variables: HR <int>,
## #   RBI <int>, SB <int>, CS <int>, BB <int>, SO <int>, IBB <int>,
## #   HBP <int>, SH <int>, SF <int>, GIDP <int>
```

Subsetting/Manipulating Data

`filter()` - subset **rows**

- Use `filter()` to obtain only PIT data

```
filter(Batting, teamID == "PIT")
```

```
## # A tibble: 4,667 × 22
##    playerID yearID stint teamID   lgID     G    AB     R     H   X2B   X3B
##       <chr>  <int> <int> <fctr> <fctr> <int> <int> <int> <int> <int> <int>
## 1 barklsa01   1887     1    PIT     NL    89   340    44    76    10     4
## 2 beeched01   1887     1    PIT     NL    41   169    15    41     8     0
## 3 bishobi01   1887     1    PIT     NL     3     9     0     0     0     0
## 4 brownto01   1887     1    PIT     NL    47   192    30    47     3     4
## 5 carrofr01   1887     1    PIT     NL   102   421    71   138    24    15
## # ... with 4,662 more rows, and 11 more variables: HR <int>, RBI <int>,
## #   SB <int>, CS <int>, BB <int>, SO <int>, IBB <int>, HBP <int>,
## #   SH <int>, SF <int>, GIDP <int>
```

Subsetting/Manipulating Data

`filter()` - subset **rows**

- Multiple filters

```
filter(Batting, teamID == "PIT" & yearID == 2000)
```

```
## # A tibble: 46 × 22
##    playerID yearID stint teamID   lgID     G    AB     R     H   X2B   X3B
##       <chr>  <int> <int> <fctr> <fctr> <int> <int> <int> <int> <int> <int>
## 1 anderji02   2000     1    PIT     NL    27    50     5     7     1     0
## 2 arroybr01   2000     1    PIT     NL    21    21     2     3     2     0
## 3  avenbr01   2000     1    PIT     NL    72   148    18    37    11     0
## 4 benjami01   2000     1    PIT     NL    93   233    28    63    18     2
## 5 bensokr01   2000     1    PIT     NL    32    65     3     6     2     0
## # ... with 41 more rows, and 11 more variables: HR <int>, RBI <int>,
## #   SB <int>, CS <int>, BB <int>, SO <int>, IBB <int>, HBP <int>,
## #   SH <int>, SF <int>, GIDP <int>
```

Subsetting/Manipulating Data

`arrange()` - reorder **rows**

```
#reorder by teamID
arrange(Batting, teamID)
```

```
## # A tibble: 101,332 × 22
##    playerID yearID stint teamID   lgID     G    AB     R     H   X2B   X3B
##       <chr>  <int> <int> <fctr> <fctr> <int> <int> <int> <int> <int> <int>
## 1 berrych01   1884     1    ALT     UA     7    25     2     6     0     0
## 2 brownji01   1884     1    ALT     UA    21    88    12    22     2     2
## 3 carropa01   1884     1    ALT     UA    11    49     4    13     1     0
## 4 connojo01   1884     1    ALT     UA     3    11     0     1     0     0
## 5 crosscl01   1884     1    ALT     UA     2     7     1     4     1     0
## # ... with 1.013e+05 more rows, and 11 more variables: HR <int>,
## #   RBI <int>, SB <int>, CS <int>, BB <int>, SO <int>, IBB <int>,
## #   HBP <int>, SH <int>, SF <int>, GIDP <int>
```

Subsetting/Manipulating Data

`arrange()` - reorder **rows**

```
#get secondary arrangement as well
arrange(Batting, teamID, G)
```

```
## # A tibble: 101,332 × 22
##    playerID yearID stint teamID   lgID     G    AB     R     H   X2B   X3B
##       <chr>  <int> <int> <fctr> <fctr> <int> <int> <int> <int> <int> <int>
## 1 daisege01   1884     1    ALT     UA     1     4     0     0     0     0
## 2 crosscl01   1884     1    ALT     UA     2     7     1     4     1     0
## 3 manloch01   1884     1    ALT     UA     2     7     1     3     0     0
## 4 connojo01   1884     1    ALT     UA     3    11     0     1     0     0
## 5 berrych01   1884     1    ALT     UA     7    25     2     6     0     0
## # ... with 1.013e+05 more rows, and 11 more variables: HR <int>,
## #   RBI <int>, SB <int>, CS <int>, BB <int>, SO <int>, IBB <int>,
## #   HBP <int>, SH <int>, SF <int>, GIDP <int>
```

Subsetting/Manipulating Data

`arrange()` - reorder **rows**

```
#descending instead
arrange(Batting, teamID, desc(G))
```

```
## # A tibble: 101,332 × 22
##    playerID yearID stint teamID   lgID     G    AB     R     H   X2B   X3B
##       <chr>  <int> <int> <fctr> <fctr> <int> <int> <int> <int> <int> <int>
## 1 smithge01   1884     1    ALT     UA    25   108     9    34     8     1
## 2 harrifr01   1884     1    ALT     UA    24    95    10    25     2     1
## 3 doughch01   1884     1    ALT     UA    23    85     6    22     5     0
## 4 murphjo01   1884     1    ALT     UA    23    94    10    14     1     0
## 5 brownji01   1884     1    ALT     UA    21    88    12    22     2     2
## # ... with 1.013e+05 more rows, and 11 more variables: HR <int>,
## #   RBI <int>, SB <int>, CS <int>, BB <int>, SO <int>, IBB <int>,
## #   HBP <int>, SH <int>, SF <int>, GIDP <int>
```

Subsetting/Manipulating Data

Piping or Chaining

- Applying multiple functions: nesting hard to parse!
- Piping or Chaining with %>% operator helps

```
arrange(filter(Batting, teamID == "PIT"), desc(G))
```

```
## # A tibble: 4,667 × 22
##    playerID yearID stint teamID   lgID     G    AB     R     H   X2B   X3B
##       <chr>  <int> <int> <fctr> <fctr> <int> <int> <int> <int> <int> <int>
## 1 mazerbi01   1967     1    PIT     NL   163   639    62   167    25     3
## 2 bonilbo01   1989     1    PIT     NL   163   616    96   173    37    10
## 3 mazerbi01   1964     1    PIT     NL   162   601    66   161    22     8
## 4 clenddo01   1965     1    PIT     NL   162   612    89   184    32    14
## 5 mazerbi01   1966     1    PIT     NL   162   621    56   163    22     7
## # ... with 4,662 more rows, and 11 more variables: HR <int>, RBI <int>,
## #   SB <int>, CS <int>, BB <int>, SO <int>, IBB <int>, HBP <int>,
## #   SH <int>, SF <int>, GIDP <int>
```

Subsetting/Manipulating Data

Piping or Chaining

- Applying multiple functions: nesting hard to parse!
- Piping or Chaining with %>% operator helps

```
Batting %>% filter(teamID == "PIT") %>% arrange(desc(G))
```

```
## # A tibble: 4,667 × 22
##    playerID yearID stint teamID   lgID     G    AB     R     H   X2B   X3B
##       <chr>  <int> <int> <fctr> <fctr> <int> <int> <int> <int> <int> <int>
## 1 mazerbi01   1967     1    PIT     NL   163   639    62   167    25     3
## 2 bonilbo01   1989     1    PIT     NL   163   616    96   173    37    10
## 3 mazerbi01   1964     1    PIT     NL   162   601    66   161    22     8
## 4 clenddo01   1965     1    PIT     NL   162   612    89   184    32    14
## 5 mazerbi01   1966     1    PIT     NL   162   621    56   163    22     7
## # ... with 4,662 more rows, and 11 more variables: HR <int>, RBI <int>,
## #   SB <int>, CS <int>, BB <int>, SO <int>, IBB <int>, HBP <int>,
## #   SH <int>, SF <int>, GIDP <int>
```

Subsetting/Manipulating Data

Piping or Chaining

- Applying multiple functions: nesting hard to parse!
- Piping or Chaining with %>% operator helps
- If `dplyr` or `magrittr` package loaded, can use anywhere

```
a<-runif(n = 10)
a
```

```
##  [1] 0.08621019 0.80290471 0.14787124 0.18874695 0.99241232 0.48686800
##  [7] 0.86030274 0.11337676 0.16658874 0.01879956
```

Subsetting/Manipulating Data

Piping or Chaining

```
#silly example
a %>% quantile()
```

```
##         0%        25%        50%        75%       100%
## 0.01879956 0.12200038 0.17766784 0.72389553 0.99241232
```

```
a %>% quantile() %>% range()
```

```
## [1] 0.01879956 0.99241232
```

Subsetting/Manipulating Data

`select()` - subset **columns**

- Often only want select variables (saw `$` and `[ , ]`)
- `select()` function has same syntax as other `dplyr` functions!

```
#Choose a single column by name
Batting %>% select(X2B)
```

```
## # A tibble: 101,332 × 1
##     X2B
##   <int>
## 1     0
## 2     6
## 3     4
## 4    10
## 5    11
## # ... with 1.013e+05 more rows
```

Subsetting/Manipulating Data

`select()` - subset **columns**

- Many ways to select variables

```
#all columns between
Batting %>% select(X2B:HR)
```

```
## # A tibble: 101,332 × 3
##     X2B   X3B    HR
##   <int> <int> <int>
## 1     0     0     0
## 2     6     0     0
## 3     4     5     0
## 4    10     2     2
## 5    11     3     0
## # ... with 1.013e+05 more rows
```

Subsetting/Manipulating Data

`select()` - subset **columns**

- Many ways to select variables

```
#all columns containing
Batting %>% select(contains("X"))
```

```
## # A tibble: 101,332 × 2
##     X2B   X3B
##   <int> <int>
## 1     0     0
## 2     6     0
## 3     4     5
## 4    10     2
## 5    11     3
## # ... with 1.013e+05 more rows
```

Subsetting/Manipulating Data

`select()` - subset **columns**

- Many ways to select variables

```
#all columns starting with
Batting %>% select(starts_with("X"))
```

```
## # A tibble: 101,332 × 2
##     X2B   X3B
##   <int> <int>
## 1     0     0
## 2     6     0
## 3     4     5
## 4    10     2
## 5    11     3
## # ... with 1.013e+05 more rows
```

Subsetting/Manipulating Data

`select()` - subset **columns**

- Many ways to select variables

```
#all columns ending with
Batting %>% select(ends_with("ID"))
```

```
## # A tibble: 101,332 × 4
##    playerID yearID teamID   lgID
##       <chr>  <int> <fctr> <fctr>
## 1 abercda01   1871    TRO     NA
## 2  addybo01   1871    RC1     NA
## 3 allisar01   1871    CL1     NA
## 4 allisdo01   1871    WS3     NA
## 5 ansonca01   1871    RC1     NA
## # ... with 1.013e+05 more rows
```

Subsetting/Manipulating Data

`mutate()` - add newly created **column**

`transmute()` - create new variable

```
##Create an Extra Base Hits variable
Batting %>% mutate(ExtraBaseHits = X2B + X3B + HR)
```

```
## # A tibble: 101,332 × 23
##    playerID yearID stint teamID   lgID     G    AB     R     H   X2B   X3B
##       <chr>  <int> <int> <fctr> <fctr> <int> <int> <int> <int> <int> <int>
## 1 abercda01   1871     1    TRO     NA     1     4     0     0     0     0
## 2  addybo01   1871     1    RC1     NA    25   118    30    32     6     0
## 3 allisar01   1871     1    CL1     NA    29   137    28    40     4     5
## 4 allisdo01   1871     1    WS3     NA    27   133    28    44    10     2
## 5 ansonca01   1871     1    RC1     NA    25   120    29    39    11     3
## # ... with 1.013e+05 more rows, and 12 more variables: HR <int>,
## #   RBI <int>, SB <int>, CS <int>, BB <int>, SO <int>, IBB <int>,
## #   HBP <int>, SH <int>, SF <int>, GIDP <int>, ExtraBaseHits <int>
```

Subsetting/Manipulating Data

`mutate()` - add newly created **column**

`transmute()` - create new variable

```
#can't see it!
Batting %>% mutate(ExtraBaseHits = X2B + X3B + HR) %>% select(ExtraBaseHits)
```

```
## # A tibble: 101,332 × 1
##   ExtraBaseHits
##           <int>
## 1             0
## 2             6
## 3             9
## 4            14
## 5            14
## # ... with 1.013e+05 more rows
```

Subsetting/Manipulating Data

`mutate()` - add newly created **column**

`transmute()` - create new variable

```
#transmute will keep the new variable only
Batting %>% transmute(ExtraBaseHits = X2B + X3B + HR)
```

```
## # A tibble: 101,332 × 1
##   ExtraBaseHits
##           <int>
## 1             0
## 2             6
## 3             9
## 4            14
## 5            14
## # ... with 1.013e+05 more rows
```

Subsetting/Manipulating Data

`group_by()` - group **rows** by a variable

`summarise()` - apply basic function to data

- Summarization - find avg # of doubles (X2B)
- Remove `NA`'s
- `NA` = Not Available (R's missing data indicator)

```
Batting %>% summarise(AvgX2B = mean(X2B, na.rm = TRUE))
```

```
## # A tibble: 1 × 1
##     AvgX2B
##      <dbl>
## 1 6.637067
```

Subsetting/Manipulating Data

`group_by()` - group **rows** by a variable

`summarise()` - apply basic function to data

- Summarization - find avg # of doubles (X2B) *by team*

```
Batting %>% group_by(teamID) %>% summarise(AvgX2B = mean(X2B, na.rm = TRUE))
```

```
## # A tibble: 149 × 2
##   teamID   AvgX2B
##   <fctr>    <dbl>
## 1    ALT 1.764706
## 2    ANA 7.320635
## 3    ARI 6.331325
## 4    ATL 5.889756
## 5    BAL 7.047974
## # ... with 144 more rows
```

Subsetting/Manipulating Data

`left_join()`, `right_join()`, `inner_join()`, `full_join()` - combine multiple DFs

(Cite: http://rpubs.com/justmarkham/dplyr-tutorial-part-2)

```
# create two simple data frames
a <- data_frame(color = c("green", "yellow", "red"), num = 1:3)
b <- data_frame(color = c("green", "yellow", "pink"), size = c("S", "M", "L"))
```

```
a
```

```
## # A tibble: 3 × 2
##    color   num
##    <chr> <int>
## 1  green     1
## 2 yellow     2
## 3    red     3
```

```
b
```

```
## # A tibble: 3 × 2
##    color  size
##    <chr> <chr>
## 1  green     S
## 2 yellow     M
## 3   pink     L
```

Subsetting/Manipulating Data

`left_join()`, `right_join()`, `inner_join()`, `full_join()` - combine multiple DFs

- Only include observations found in both "a" and "b" (automatically joins on variables that appear in both tables)

```
inner_join(a, b)
```

```
## Joining, by = "color"
```

```
## # A tibble: 2 × 3
##    color   num  size
##    <chr> <int> <chr>
## 1  green     1     S
## 2 yellow     2     M
```

Subsetting/Manipulating Data

`left_join()`, `right_join()`, `inner_join()`, `full_join()` - combine multiple DFs

- include observations found in either "a" or "b"

```
full_join(a, b)
```

```
## Joining, by = "color"
```

```
## # A tibble: 4 × 3
##    color   num  size
##    <chr> <int> <chr>
## 1  green     1     S
## 2 yellow     2     M
## 3    red     3  <NA>
## 4   pink    NA     L
```

Subsetting/Manipulating Data

`left_join()`, `right_join()`, `inner_join()`, `full_join()` - combine multiple DFs

- include all observations found in "a"

```
left_join(a, b)
```

```
## Joining, by = "color"
```

```
## # A tibble: 3 × 3
##    color   num  size
##    <chr> <int> <chr>
## 1  green     1     S
## 2 yellow     2     M
## 3    red     3  <NA>
```

Subsetting/Manipulating Data

`left_join()`, `right_join()`, `inner_join()`, `full_join()` - combine multiple DFs

- include all observations found in "b"

```
right_join(a, b)
```

```
## Joining, by = "color"
```

```
## # A tibble: 3 × 3
##    color   num  size
##    <chr> <int> <chr>
## 1  green     1     S
## 2 yellow     2     M
## 3   pink    NA     L
```

Subsetting/Manipulating Data

`left_join()`, `right_join()`, `inner_join()`, `full_join()` - combine multiple DFs

- `right_join(a, b)` is identical to `left_join(b, a)` except for column ordering

```
left_join(b, a)
```

```
## Joining, by = "color"
```

```
## # A tibble: 3 × 3
##    color  size   num
##    <chr> <chr> <int>
## 1  green     S     1
## 2 yellow     M     2
## 3   pink     L    NA
```

Subsetting/Manipulating Data

`left_join()`, `right_join()`, `inner_join()`, `full_join()` - combine multiple DFs

- filter "a" to only show observations that match "b"

```
semi_join(a, b)
```

```
## Joining, by = "color"
```

```
## # A tibble: 2 × 2
##    color   num
##    <chr> <int>
## 1  green     1
## 2 yellow     2
```

Subsetting/Manipulating Data

`left_join()`, `right_join()`, `inner_join()`, `full_join()` - combine multiple DFs

- filter "a" to only show observations that don't match "b"

```
anti_join(a, b)
```

```
## Joining, by = "color"
```

```
## # A tibble: 1 × 2
##   color   num
##   <chr> <int>
## 1   red     3
```

Subsetting/Manipulating Data

`left_join()`, `right_join()`, `inner_join()`, `full_join()` - combine multiple DFs

- sometimes matching variables don't have identical names

```
b <- b %>% rename(col = color)
a
```

```
## # A tibble: 3 × 2
##    color   num
##    <chr> <int>
## 1  green     1
## 2 yellow     2
## 3    red     3
```

```
b
```

```
## # A tibble: 3 × 2
##      col  size
##    <chr> <chr>
## 1  green     S
## 2 yellow     M
## 3   pink     L
```

Subsetting/Manipulating Data

`left_join()`, `right_join()`, `inner_join()`, `full_join()` - combine multiple DFs

- specify that the join should occur by matching "color" in "a" with "col" in "b"

```
inner_join(a, b, by = c("color" = "col"))
```

```
## # A tibble: 2 × 3
##    color   num  size
##    <chr> <int> <chr>
## 1  green     1     S
## 2 yellow     2     M
```

Subsetting/Manipulating Data

Overview of **`dplyr`** package cheatsheet

- Basic commands
  - `tbl_df()` - convert data frame to one with better printing
  - `filter()` - subset **rows**
  - `arrange()` - reorder **rows**
  - `select()` - subset **columns**
  - `mutate()` - add newly created **column**
  - `transmute()` - create new variable
  - `group_by()` - group **rows** by a variable
  - `summarise()` - apply basic function to data
  - `left_join()`, `right_join()`, `inner_join()`, `full_join()` - commands to combine multiple dfs

Manipulating Data

`tidyr` package

Easily allows for two very important actions

- **gather()** - takes multiple columns, and gathers them into key-value pairs
  - Make wide data longer
  - Most important as analysis methods often prefer this form
- **spread()** - takes two columns (key & value) and spreads in to multiple columns
  - Make "long" data wider

Manipulating Data

tidyr package

- Data in 'Wide' form

```
tempsData <- read_delim(file = "https://raw.githubusercontent.com/jbpost2/Programming-in-R/
                        master/datasets/cityTemps.txt", delim = " ")
tempsData
```

```
## Parsed with column specification:
## cols(
##   city = col_character(),
##   sun = col_integer(),
##   mon = col_integer(),
##   tue = col_integer(),
##   wed = col_integer(),
##   thr = col_integer(),
##   fri = col_integer(),
##   sat = col_integer()
## )
```

Manipulating Data

tidyr package

- Switch to 'Long' form with `gather()` (see help)
- Can provide columns to `gather()` in many ways!

```
gather(tempsData, key = day, value = temp, 2:8)
```

```
## # A tibble: 42 × 3
##        city   day  temp
##       <chr> <chr> <int>
## 1   atlanta   sun    81
## 2 baltimore   sun    73
## 3 charlotte   sun    82
## 4    denver   sun    72
## 5 ellington   sun    51
## # ... with 37 more rows
```

Manipulating Data

tidyr package

- Switch to 'Long' form with `gather()` (see help)
- Can provide columns to `gather()` in many ways!

```
newTempsData<-gather(tempsData, key = day, value = temp, sun,
mon, tue, wed, thr, fri, sat)
```

```
## # A tibble: 42 × 3
##        city   day  temp
##       <chr> <chr> <int>
## 1   atlanta   sun    81
## 2 baltimore   sun    73
## 3 charlotte   sun    82
## 4    denver   sun    72
## 5 ellington   sun    51
## # ... with 37 more rows
```

Manipulating Data

tidyr package

- Switch to 'Wide' form with `spread()` (see help)

```
spread(newTempsData, key = day, value = temp)
```

```
## # A tibble: 6 × 8
##       city   fri   mon   sat   sun   thr   tue   wed
## *     <chr> <int> <int> <int> <int> <int> <int> <int>
## 1   atlanta    91    87    94    81    88    83    79
## 2 baltimore    75    75    79    73    73    70    78
## 3 charlotte    88    80    93    82    83    75    82
## 4    denver    71    71    58    72    72    67    68
## 5 ellington    56    42    59    51    55    47    52
## 6 frankfort    74    70    79    70    74    72    70
```

Recap!

- Tidyverse useful
- `dplyr` to manipulate data
- `tidyr` to expand, condense data

Activity

- **Manipulating Data Activity** instructions available on web
- Work in small groups
- Ask questions! TAs and I will float about the room
- Feel free to ask questions about anything you didn't understand as well!

What do we want to be able to do?

- Restructure Data/Clean Data
- **Streamline repeated sections of code**
- Improve efficiency of code
- Write custom functions to simplify code

For Loops

- Idea:
    - Run code repeatedly
    - Often change something as well
- Syntax

```
for(index in values){
  code to be run
}
```

For Loops

- index defines 'counter' or variable that varies

```
for (i in 1:10){
  print(i)
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
## [1] 8
## [1] 9
## [1] 10
```

For Loops

- 'values' define which values index takes on

```
for (i in 1:10){
  print(i)
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
## [1] 8
## [1] 9
## [1] 10
```

For Loops

- 'values' define which values index takes on

```
for (value in c("cat","hat","worm")){
  print(value)
}
```

```
## [1] "cat"
## [1] "hat"
## [1] "worm"
```

For Loops

- Code in loop can change based on index
- Create small data set

```
set.seed(10)
data<-round(runif(5),2)
data
```

```
## [1] 0.51 0.31 0.43 0.69 0.09
```

For Loops

- Code in loop can change based on index

```
words<-c("first", "second", "third", "fourth", "fifth")
```

- Loop through and print out the phrase

"The (#ed) data point is (# from data vector)."

```
paste0("The ", words[1], " data point is ", data[1], ".")
```

```
## [1] "The first data point is 0.51."
```

For Loops

- Code in loop can change based on index

```
for (i in 1:5){
  print(paste0("The ", words[i], " data point is ",
               data[i], "."))
}
```

```
## [1] "The first data point is 0.51."
## [1] "The second data point is 0.31."
## [1] "The third data point is 0.43."
## [1] "The fourth data point is 0.69."
## [1] "The fifth data point is 0.09."
```

For Loops

- Example: Find `summary()` for each column of a data set
- Could loop through numeric columns
- Find `summary()` for each

- Consider smaller batting data set

```
Batting2010 <- Batting %>% filter(yearID == 2010) %>%
  select(playerID, teamID, G, AB, R, H, X2B, X3B, HR)
```

For Loops

- Want to find `summary()` for each column of a data set

```
summary(Batting2010[ , 3])
```

```
##        G
##  Min.   :  1.00
##  1st Qu.: 15.00
##  Median : 33.00
##  Mean   : 50.83
##  3rd Qu.: 74.00
##  Max.   :162.00
```

For Loops

- Loop through numeric columns

```
stats <- matrix(nrow = 6, ncol = 7)
for (i in 1:(dim(Batting2010)[2] - 2)){
  stats[ , i]<-summary(Batting2010[ , i + 2])
}
stats
```

```
##      [,1]              [,2]              [,3]
## [1,] "Min.   :  1.00  " "Min.   :  0.0  " "Min.   :  0.00  "
## [2,] "1st Qu.: 15.00  " "1st Qu.:  0.0  " "1st Qu.:  0.00  "
## [3,] "Median : 33.00  " "Median : 24.5  " "Median :  2.00  "
## [4,] "Mean   : 50.83  " "Mean   :121.9  " "Mean   : 15.71  "
## [5,] "3rd Qu.: 74.00  " "3rd Qu.:186.0  " "3rd Qu.: 22.00  "
## [6,] "Max.   :162.00  " "Max.   :680.0  " "Max.   :115.00  "
##      [,4]              [,5]               [,6]
## [1,] "Min.   :  0.00  " "Min.   : 0.000  " "Min.   : 0.0000  "
## [2,] "1st Qu.:  0.00  " "1st Qu.: 0.000  " "1st Qu.: 0.0000  "
## [3,] "Median :  4.00  " "Median : 0.000  " "Median : 0.0000  "
## [4,] "Mean   : 31.38  " "Mean   : 6.258  " "Mean   : 0.6386  "
```

For Loops

- Add column names

```
colnames(stats) <- names(Batting2010)[3:9]
stats
```

```
##      G                  AB                 R
## [1,] "Min.   :  1.00  " "Min.   :  0.0  " "Min.   :  0.00  "
## [2,] "1st Qu.: 15.00  " "1st Qu.:  0.0  " "1st Qu.:  0.00  "
## [3,] "Median : 33.00  " "Median : 24.5  " "Median :  2.00  "
## [4,] "Mean   : 50.83  " "Mean   :121.9  " "Mean   : 15.71  "
## [5,] "3rd Qu.: 74.00  " "3rd Qu.:186.0  " "3rd Qu.: 22.00  "
## [6,] "Max.   :162.00  " "Max.   :680.0  " "Max.   :115.00  "
##      H                  X2B                X3B
## [1,] "Min.   :  0.00  " "Min.   : 0.000  " "Min.   : 0.0000  "
## [2,] "1st Qu.:  0.00  " "1st Qu.: 0.000  " "1st Qu.: 0.0000  "
## [3,] "Median :  4.00  " "Median : 0.000  " "Median : 0.0000  "
## [4,] "Mean   : 31.38  " "Mean   : 6.258  " "Mean   : 0.6386  "
## [5,] "3rd Qu.: 45.00  " "3rd Qu.: 8.000  " "3rd Qu.: 1.0000  "
## [6,] "Max.   :214.00  " "Max.   :49.000  " "Max.   :14.0000  "
##      HR
```

Vectorized Function

- Much better way to do this type of thing
- Loops are slow, didn't keep attributes here
- Covered later today!

Recap!

- **For loops** reduce redundant code
- Syntax

```
for (index in values){
  code to execute
}
```

- Values can be a sequence of numbers or character values
- Not ideal in R

Activity

- **For Loops Activity** instructions available on web
- Work in small groups
- Ask questions! TAs and I will float about the room
- Feel free to ask questions about anything you didn't understand as well!