More on dplyr

Austin Chiang

October 30, 2017

More on DPLYR

Introduction

With data analysis becoming more and more prominent in so many aspects of life today, it becomes extremely important to work with data that is properly organized so that it becomes easier to work with. However, rarely will we be able to get our datasets exactly the way we want. As a result, we need to rely on data manipulation in order to do things like add variables or reorganize the columns of data frames in order to better suit the user and what he or she is trying to achieve with the information. We learned in class that one way to transform data is to use the *dplyr* package. This package gives us access to many functions that will allow us to better organize our data and solve many of the challenges that we will face when it comes to data manipulation. In this post we will be working with data about the rushing offense of NFL teams in order to fully understand the functions of the *dplyr* package. I chose *dplyr* as the main topic of this post because I found data manipulation to be quite an interesting and useful tool when it came to better understanding data. I am an avid basketball fan so being able to use the data manipulation tools of the package in order to see the data in a new light was just something that I wanted to learn more about and share with others in this post. I hope you all enjoy it.

```
library(dplyr)
```

```
## Warning: package 'dplyr' was built under R version 3.4.2
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
file <- "nfl_team_rushing_offense.csv"
team_rushing <- read.csv(file, stringsAsFactors = FALSE)
team_rushing #Rushing Offense Data from the 2016 NFL season for all teams.
```

```
##    Rk                  Tm  G Att  Yds TD Lng Y.A   Y.G Fmb    EXP
## 1   1        Buffalo Bills 16 492 2630 29  75 5.3 164.4  15  59.91
## 2   2       Dallas Cowboys 16 499 2396 24  60 4.8 149.8  22  39.96
## 3   3     Tennessee Titans 16 476 2187 16  75 4.6 136.7  16  -4.29
## 4   4  San Francisco 49ers 16 458 2019 15  47 4.4 126.2  29 -16.25
## 5   5      Atlanta Falcons 16 421 1928 20  75 4.6 120.5   8  20.78
## 6   6      Oakland Raiders 16 434 1922 17  75 4.4 120.1  17 -38.09
## 7   7 New England Patriots 16 482 1872 19  44 3.9 117.0  27 -36.10
## 8   8       Houston Texans 16 456 1859  8  45 4.1 116.2  20 -39.31
## 9   9       Miami Dolphins 16 406 1824 14  62 4.5 114.0  26 -29.06
## 10 10    Carolina Panthers 16 453 1814 16  47 4.0 113.4  13 -24.98
## 11 11  Philadelphia Eagles 16 438 1813 16  30 4.1 113.3  24 -17.58
## 12 12        New York Jets 16 418 1802 10  35 4.3 112.6  24  -8.50
## 13 13   Cincinnati Bengals 16 446 1769 17  74 4.0 110.6  20   1.28
## 14 14  Pittsburgh Steelers 16 409 1760 13  60 4.3 110.0  15 -16.84
## 15 15   Kansas City Chiefs 16 412 1748 15  70 4.2 109.3  18 -24.46
## 16 16   New Orleans Saints 16 404 1742 17  75 4.3 108.9  17  15.68
## 17 17        Chicago Bears 16 380 1735 10  69 4.6 108.4  26 -19.89
## 18 18    Arizona Cardinals 16 399 1732 20  58 4.3 108.3  27 -14.69
## 19 19     Cleveland Browns 16 350 1712 13  85 4.9 107.0  26 -15.44
## 20 20    Green Bay Packers 16 374 1701 11  61 4.5 106.3  21  -3.65
## 21 21  Washington Redskins 16 379 1696 17  66 4.5 106.0  19  -6.19
## 22 22 Jacksonville Jaguars 16 392 1631  8  57 4.2 101.9  23 -44.63
## 23 23   Indianapolis Colts 16 409 1628 13  33 4.0 101.8  16  -4.97
## 24 24 Tampa Bay Buccaneers 16 453 1616  8  45 3.6 101.0  19 -68.53
## 25 25     Seattle Seahawks 16 403 1591 13  75 3.9  99.4  22 -33.05
## 26 26   San Diego Chargers 16 398 1510 10  48 3.8  94.4  22 -32.24
## 27 27       Denver Broncos 16 410 1484 11  64 3.6  92.8  24 -57.61
## 28 28     Baltimore Ravens 16 367 1463 10  41 4.0  91.4  19 -11.25
## 29 29      New York Giants 16 398 1412  6  25 3.5  88.3  22 -60.21
## 30 30        Detroit Lions 16 350 1310  9  42 3.7  81.9   9 -17.32
## 31 31     Los Angeles Rams 16 375 1252  9  30 3.3  78.3  24 -54.51
## 32 32    Minnesota Vikings 16 380 1205  9  36 3.2  75.3  20 -78.99
## 33 NA                     NA  NA   NA NA  NA  NA    NA  NA     NA
## 34 NA                     NA  NA   NA NA  NA  NA    NA  NA     NA
## 35 NA                     NA  NA   NA NA  NA  NA    NA  NA     NA
```

Basics

There are many functions in the *dplyr* package but we will only cover a couple of key functions in this post. These functions will allow us to manipulate data in order to have an easier time with analyzing it. We will use these functions on our nfl rushing offense data to show exactly what they do.

- 'filter()' allows us to extract observations that meet logical criteria.
- 'arrange()' allows us to reorder the rows of the data frame.
- 'select()' allows us to choose variables by name or a helper function.
- 'mutate()' allows us to create new variables in the data frame.
- 'summarize()' allows us to collapse the data into a single summary.

All these functions take in a data frame as the first argument. The following arguments describe thigns to do with the data frame that we passed in. All of them also return a data frame.

Filtering Rows

The function 'filter()' allows you to subset observations based on values that meet a certain logical criteria. For example, we can use 'filter()' in order to select all the teams that rush for more than one hundred yards per game.

```
filter(team_rushing, Y.G > 100)
```

```
##    Rk                  Tm  G Att  Yds TD Lng Y.A   Y.G Fmb    EXP
## 1   1        Buffalo Bills 16 492 2630 29  75 5.3 164.4  15  59.91
## 2   2       Dallas Cowboys 16 499 2396 24  60 4.8 149.8  22  39.96
## 3   3     Tennessee Titans 16 476 2187 16  75 4.6 136.7  16  -4.29
## 4   4  San Francisco 49ers 16 458 2019 15  47 4.4 126.2  29 -16.25
## 5   5      Atlanta Falcons 16 421 1928 20  75 4.6 120.5   8  20.78
## 6   6      Oakland Raiders 16 434 1922 17  75 4.4 120.1  17 -38.09
## 7   7 New England Patriots 16 482 1872 19  44 3.9 117.0  27 -36.10
## 8   8       Houston Texans 16 456 1859  8  45 4.1 116.2  20 -39.31
## 9   9       Miami Dolphins 16 406 1824 14  62 4.5 114.0  26 -29.06
## 10 10    Carolina Panthers 16 453 1814 16  47 4.0 113.4  13 -24.98
## 11 11  Philadelphia Eagles 16 438 1813 16  30 4.1 113.3  24 -17.58
## 12 12        New York Jets 16 418 1802 10  35 4.3 112.6  24  -8.50
## 13 13   Cincinnati Bengals 16 446 1769 17  74 4.0 110.6  20   1.28
## 14 14  Pittsburgh Steelers 16 409 1760 13  60 4.3 110.0  15 -16.84
## 15 15   Kansas City Chiefs 16 412 1748 15  70 4.2 109.3  18 -24.46
## 16 16   New Orleans Saints 16 404 1742 17  75 4.3 108.9  17  15.68
## 17 17        Chicago Bears 16 380 1735 10  69 4.6 108.4  26 -19.89
## 18 18    Arizona Cardinals 16 399 1732 20  58 4.3 108.3  27 -14.69
## 19 19     Cleveland Browns 16 350 1712 13  85 4.9 107.0  26 -15.44
## 20 20    Green Bay Packers 16 374 1701 11  61 4.5 106.3  21  -3.65
## 21 21  Washington Redskins 16 379 1696 17  66 4.5 106.0  19  -6.19
## 22 22 Jacksonville Jaguars 16 392 1631  8  57 4.2 101.9  23 -44.63
## 23 23   Indianapolis Colts 16 409 1628 13  33 4.0 101.8  16  -4.97
## 24 24 Tampa Bay Buccaneers 16 453 1616  8  45 3.6 101.0  19 -68.53
```

When the code is run, the filtering operation is executed and it returns a new data frame with values that match the logical criteria passed in in the arguments. However, the original data frame is not changed so if we want to save the value, we need to assign the data frame to a new variable.

```
team_rushing_over_100 <- filter(team_rushing, Y.G > 100)
team_rushing_over_100
```

```
##    Rk                  Tm  G Att  Yds TD Lng Y.A   Y.G Fmb    EXP
## 1   1        Buffalo Bills 16 492 2630 29  75 5.3 164.4  15  59.91
## 2   2       Dallas Cowboys 16 499 2396 24  60 4.8 149.8  22  39.96
## 3   3     Tennessee Titans 16 476 2187 16  75 4.6 136.7  16  -4.29
## 4   4  San Francisco 49ers 16 458 2019 15  47 4.4 126.2  29 -16.25
## 5   5      Atlanta Falcons 16 421 1928 20  75 4.6 120.5   8  20.78
## 6   6      Oakland Raiders 16 434 1922 17  75 4.4 120.1  17 -38.09
## 7   7 New England Patriots 16 482 1872 19  44 3.9 117.0  27 -36.10
## 8   8       Houston Texans 16 456 1859  8  45 4.1 116.2  20 -39.31
## 9   9       Miami Dolphins 16 406 1824 14  62 4.5 114.0  26 -29.06
## 10 10    Carolina Panthers 16 453 1814 16  47 4.0 113.4  13 -24.98
## 11 11  Philadelphia Eagles 16 438 1813 16  30 4.1 113.3  24 -17.58
## 12 12        New York Jets 16 418 1802 10  35 4.3 112.6  24  -8.50
## 13 13   Cincinnati Bengals 16 446 1769 17  74 4.0 110.6  20   1.28
## 14 14  Pittsburgh Steelers 16 409 1760 13  60 4.3 110.0  15 -16.84
## 15 15   Kansas City Chiefs 16 412 1748 15  70 4.2 109.3  18 -24.46
## 16 16   New Orleans Saints 16 404 1742 17  75 4.3 108.9  17  15.68
## 17 17        Chicago Bears 16 380 1735 10  69 4.6 108.4  26 -19.89
## 18 18    Arizona Cardinals 16 399 1732 20  58 4.3 108.3  27 -14.69
## 19 19     Cleveland Browns 16 350 1712 13  85 4.9 107.0  26 -15.44
## 20 20    Green Bay Packers 16 374 1701 11  61 4.5 106.3  21  -3.65
## 21 21  Washington Redskins 16 379 1696 17  66 4.5 106.0  19  -6.19
## 22 22 Jacksonville Jaguars 16 392 1631  8  57 4.2 101.9  23 -44.63
## 23 23   Indianapolis Colts 16 409 1628 13  33 4.0 101.8  16  -4.97
## 24 24 Tampa Bay Buccaneers 16 453 1616  8  45 3.6 101.0  19 -68.53
```

Comparison Operators

The following are comparison operators that allow us to use 'filter()' to its fullest. They enable us to properly select the observations that we desire so that we get the the proper data frame back: > (greater than), >= (greater than or equal to), < (less than), <= (less than or equal to), != (not equal to), == (equal).

Logical Operators

The following image shows types of logical operators that we can use to help obtain the observations that we want when using the filter function.

```
library(imager)
```

```
## Warning: package 'imager' was built under R version 3.4.2
```

```
## Loading required package: plyr
```

```
## ------------------------------------------------------------------------------
```

```
## You have loaded plyr after dplyr - this is likely to cause problems.
## If you need functions from both plyr and dplyr, please load plyr first, then dplyr:
## library(plyr); library(dplyr)
```

```
## ------------------------------------------------------------------------------
```

```
##
## Attaching package: 'plyr'
```

```
## The following objects are masked from 'package:dplyr':
##
##     arrange, count, desc, failwith, id, mutate, rename, summarise,
##     summarize
```

```
## Loading required package: magrittr
```

```
##
## Attaching package: 'imager'
```

```
## The following object is masked from 'package:magrittr':
##
##     add
```

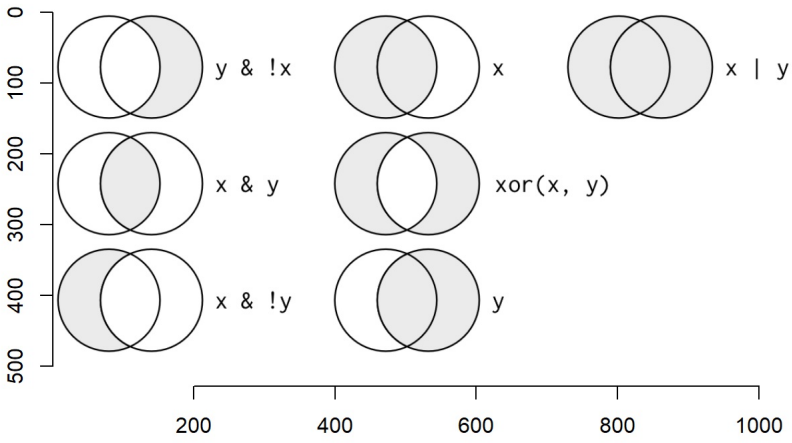
```
## The following object is masked from 'package:plyr':
##
##     liply
```

```
## The following objects are masked from 'package:stats':
##
##     convolve, spectrum
```

```
## The following object is masked from 'package:graphics':
##
##     frame
```

```
## The following object is masked from 'package:base':
##
##     save.image
```

```
pp <- load.image("Logical_Operators.png")
plot(pp)
```



Arranging Rows

'arrange()' allows us to alter the order of our data frame. For example, we can use the arrange function in order to order our team_rushing data frame by the number of rushing attempts by each team in increasing order.

```
arrange(team_rushing, Att) #orders the data frame by total rushing attempts.
```

```
##    Rk                  Tm  G Att  Yds TD Lng Y.A   Y.G Fmb    EXP
## 1  19     Cleveland Browns 16 350 1712 13  85 4.9 107.0  26 -15.44
## 2  30        Detroit Lions 16 350 1310  9  42 3.7  81.9   9 -17.32
## 3  28     Baltimore Ravens 16 367 1463 10  41 4.0  91.4  19 -11.25
## 4  20    Green Bay Packers 16 374 1701 11  61 4.5 106.3  21  -3.65
## 5  31     Los Angeles Rams 16 375 1252  9  30 3.3  78.3  24 -54.51
## 6  21  Washington Redskins 16 379 1696 17  66 4.5 106.0  19  -6.19
## 7  17        Chicago Bears 16 380 1735 10  69 4.6 108.4  26 -19.89
## 8  32    Minnesota Vikings 16 380 1205  9  36 3.2  75.3  20 -78.99
## 9  22 Jacksonville Jaguars 16 392 1631  8  57 4.2 101.9  23 -44.63
## 10 26   San Diego Chargers 16 398 1510 10  48 3.8  94.4  22 -32.24
## 11 29      New York Giants 16 398 1412  6  25 3.5  88.3  22 -60.21
## 12 18    Arizona Cardinals 16 399 1732 20  58 4.3 108.3  27 -14.69
## 13 25     Seattle Seahawks 16 403 1591 13  75 3.9  99.4  22 -33.05
## 14 16   New Orleans Saints 16 404 1742 17  75 4.3 108.9  17  15.68
## 15  9       Miami Dolphins 16 406 1824 14  62 4.5 114.0  26 -29.06
## 16 14  Pittsburgh Steelers 16 409 1760 13  60 4.3 110.0  15 -16.84
## 17 23   Indianapolis Colts 16 409 1628 13  33 4.0 101.8  16  -4.97
## 18 27       Denver Broncos 16 410 1484 11  64 3.6  92.8  24 -57.61
## 19 15   Kansas City Chiefs 16 412 1748 15  70 4.2 109.3  18 -24.46
## 20 12        New York Jets 16 418 1802 10  35 4.3 112.6  24  -8.50
## 21  5      Atlanta Falcons 16 421 1928 20  75 4.6 120.5   8  20.78
## 22  6      Oakland Raiders 16 434 1922 17  75 4.4 120.1  17 -38.09
## 23 11  Philadelphia Eagles 16 438 1813 16  30 4.1 113.3  24 -17.58
## 24 13   Cincinnati Bengals 16 446 1769 17  74 4.0 110.6  20   1.28
## 25 10    Carolina Panthers 16 453 1814 16  47 4.0 113.4  13 -24.98
## 26 24 Tampa Bay Buccaneers 16 453 1616  8  45 3.6 101.0  19 -68.53
## 27  8       Houston Texans 16 456 1859  8  45 4.1 116.2  20 -39.31
## 28  4  San Francisco 49ers 16 458 2019 15  47 4.4 126.2  29 -16.25
## 29  3     Tennessee Titans 16 476 2187 16  75 4.6 136.7  16  -4.29
## 30  7 New England Patriots 16 482 1872 19  44 3.9 117.0  27 -36.10
## 31  1        Buffalo Bills 16 492 2630 29  75 5.3 164.4  15  59.91
## 32  2       Dallas Cowboys 16 499 2396 24  60 4.8 149.8  22  39.96
## 33 NA                      NA  NA   NA NA  NA  NA    NA  NA     NA
## 34 NA                      NA  NA   NA NA  NA  NA    NA  NA     NA
## 35 NA                      NA  NA   NA NA  NA  NA    NA  NA     NA
```

We could use the function 'desc()' on the variable we which to arrange the data frame by in order to return the data frame arranged by the variable in descending order instead.

```
arrange(team_rushing, desc(Att)) #arranges team_rushing by total rushing attempts in descending order.
```

```
##    Rk                  Tm  G Att  Yds TD Lng Y.A   Y.G Fmb    EXP
## 1   2       Dallas Cowboys 16 499 2396 24  60 4.8 149.8  22  39.96
## 2   1        Buffalo Bills 16 492 2630 29  75 5.3 164.4  15  59.91
## 3   7 New England Patriots 16 482 1872 19  44 3.9 117.0  27 -36.10
## 4   3     Tennessee Titans 16 476 2187 16  75 4.6 136.7  16  -4.29
## 5   4  San Francisco 49ers 16 458 2019 15  47 4.4 126.2  29 -16.25
## 6   8       Houston Texans 16 456 1859  8  45 4.1 116.2  20 -39.31
## 7  10    Carolina Panthers 16 453 1814 16  47 4.0 113.4  13 -24.98
## 8  24 Tampa Bay Buccaneers 16 453 1616  8  45 3.6 101.0  19 -68.53
## 9  13   Cincinnati Bengals 16 446 1769 17  74 4.0 110.6  20   1.28
## 10 11  Philadelphia Eagles 16 438 1813 16  30 4.1 113.3  24 -17.58
## 11  6      Oakland Raiders 16 434 1922 17  75 4.4 120.1  17 -38.09
## 12  5      Atlanta Falcons 16 421 1928 20  75 4.6 120.5   8  20.78
## 13 12        New York Jets 16 418 1802 10  35 4.3 112.6  24  -8.50
## 14 15   Kansas City Chiefs 16 412 1748 15  70 4.2 109.3  18 -24.46
## 15 27       Denver Broncos 16 410 1484 11  64 3.6  92.8  24 -57.61
## 16 14  Pittsburgh Steelers 16 409 1760 13  60 4.3 110.0  15 -16.84
## 17 23   Indianapolis Colts 16 409 1628 13  33 4.0 101.8  16  -4.97
## 18  9       Miami Dolphins 16 406 1824 14  62 4.5 114.0  26 -29.06
## 19 16   New Orleans Saints 16 404 1742 17  75 4.3 108.9  17  15.68
## 20 25     Seattle Seahawks 16 403 1591 13  75 3.9  99.4  22 -33.05
## 21 18    Arizona Cardinals 16 399 1732 20  58 4.3 108.3  27 -14.69
## 22 26   San Diego Chargers 16 398 1510 10  48 3.8  94.4  22 -32.24
## 23 29      New York Giants 16 398 1412  6  25 3.5  88.3  22 -60.21
## 24 22 Jacksonville Jaguars 16 392 1631  8  57 4.2 101.9  23 -44.63
## 25 17        Chicago Bears 16 380 1735 10  69 4.6 108.4  26 -19.89
## 26 32    Minnesota Vikings 16 380 1205  9  36 3.2  75.3  20 -78.99
## 27 21  Washington Redskins 16 379 1696 17  66 4.5 106.0  19  -6.19
## 28 31     Los Angeles Rams 16 375 1252  9  30 3.3  78.3  24 -54.51
## 29 20    Green Bay Packers 16 374 1701 11  61 4.5 106.3  21  -3.65
## 30 28     Baltimore Ravens 16 367 1463 10  41 4.0  91.4  19 -11.25
## 31 19     Cleveland Browns 16 350 1712 13  85 4.9 107.0  26 -15.44
## 32 30        Detroit Lions 16 350 1310  9  42 3.7  81.9   9 -17.32
## 33 NA                     NA  NA   NA NA  NA  NA    NA  NA     NA
## 34 NA                     NA  NA   NA NA  NA  NA    NA  NA     NA
## 35 NA                     NA  NA   NA NA  NA  NA    NA  NA     NA
```

Using the arrange function allows us to easily see the maximums and minimums for each variable in the data frame. This can be a powerful tool in analyzing any kind of dataset.

Selecting Columns

Although our data frame of the rushing offense for every team in the NFL for the 2016 season is quite small in terms of the number of variables, ‘select()’ is still a useful tool because most datasets can have up to thousands of distinct variables to look at. IT is here where the select function shines. It allows us to basically focus our attention on a selection of variables by using operations on the names of the variables.

If we just wanted to see the team names and the number of rushing touchdowns they scored in the 2016 season, we could use ‘select()’ to get the data frame that we are looking for.

```
select(team_rushing, Tm, TD)
```

```
##                     Tm TD
## 1         Buffalo Bills 29
## 2        Dallas Cowboys 24
## 3      Tennessee Titans 16
## 4   San Francisco 49ers 15
## 5       Atlanta Falcons 20
## 6       Oakland Raiders 17
## 7  New England Patriots 19
## 8        Houston Texans  8
## 9        Miami Dolphins 14
## 10    Carolina Panthers 16
## 11  Philadelphia Eagles 16
## 12        New York Jets 10
## 13   Cincinnati Bengals 17
## 14  Pittsburgh Steelers 13
## 15   Kansas City Chiefs 15
## 16   New Orleans Saints 17
## 17        Chicago Bears 10
## 18    Arizona Cardinals 20
## 19     Cleveland Browns 13
## 20    Green Bay Packers 11
## 21  Washington Redskins 17
## 22 Jacksonville Jaguars  8
## 23   Indianapolis Colts 13
## 24 Tampa Bay Buccaneers  8
## 25     Seattle Seahawks 13
## 26   San Diego Chargers 10
## 27       Denver Broncos 11
## 28     Baltimore Ravens 10
## 29      New York Giants  6
## 30        Detroit Lions  9
## 31     Los Angeles Rams  9
## 32    Minnesota Vikings  9
## 33                     NA
## 34                     NA
## 35                     NA
```

Or if we wanted the Team, total yards gained, yards per game, and yards per attempt.

```
select(team_rushing, Tm, Yds, Y.G, Y.A)
```

```
##                     Tm  Yds   Y.G Y.A
## 1         Buffalo Bills 2630 164.4 5.3
## 2        Dallas Cowboys 2396 149.8 4.8
## 3      Tennessee Titans 2187 136.7 4.6
## 4   San Francisco 49ers 2019 126.2 4.4
## 5       Atlanta Falcons 1928 120.5 4.6
## 6       Oakland Raiders 1922 120.1 4.4
## 7  New England Patriots 1872 117.0 3.9
## 8        Houston Texans 1859 116.2 4.1
## 9        Miami Dolphins 1824 114.0 4.5
## 10    Carolina Panthers 1814 113.4 4.0
## 11  Philadelphia Eagles 1813 113.3 4.1
## 12        New York Jets 1802 112.6 4.3
## 13   Cincinnati Bengals 1769 110.6 4.0
## 14  Pittsburgh Steelers 1760 110.0 4.3
## 15   Kansas City Chiefs 1748 109.3 4.2
## 16   New Orleans Saints 1742 108.9 4.3
## 17        Chicago Bears 1735 108.4 4.6
## 18    Arizona Cardinals 1732 108.3 4.3
## 19     Cleveland Browns 1712 107.0 4.9
## 20    Green Bay Packers 1701 106.3 4.5
## 21  Washington Redskins 1696 106.0 4.5
## 22 Jacksonville Jaguars 1631 101.9 4.2
## 23   Indianapolis Colts 1628 101.8 4.0
## 24 Tampa Bay Buccaneers 1616 101.0 3.6
## 25     Seattle Seahawks 1591  99.4 3.9
## 26   San Diego Chargers 1510  94.4 3.8
## 27       Denver Broncos 1484  92.8 3.6
## 28     Baltimore Ravens 1463  91.4 4.0
## 29      New York Giants 1412  88.3 3.5
## 30        Detroit Lions 1310  81.9 3.7
## 31     Los Angeles Rams 1252  78.3 3.3
## 32    Minnesota Vikings 1205  75.3 3.2
## 33                     NA    NA  NA
## 34                     NA    NA  NA
## 35                     NA    NA  NA
```

Mutate()

In addition to selecting and arranging the data in the data frame, another useful tool is the ability to actually alter the data by adding new variables to the data frame in order to look at the data in a new way. The function will always add the new variable to the end of your data frame.

However, it is important to note that mutate will not save the changes to the current data frame. In order to save the cahnges, we will need to assign the new data frame to the name of the data frame or if we want a new data frame object to be made, a new variable.

Suppose that we want to see the number of rushing touchdowns per game that each team scores. Well, we can use the mutate function in order to create a new column in our team_rushing data frame in order to to achieve this.

```
mutate(team_rushing, TD.G = TD / G)
```

```
##    Rk                  Tm  G Att  Yds TD Lng Y.A   Y.G Fmb    EXP   TD.G
## 1   1        Buffalo Bills 16 492 2630 29  75 5.3 164.4  15  59.91 1.8125
## 2   2       Dallas Cowboys 16 499 2396 24  60 4.8 149.8  22  39.96 1.5000
## 3   3     Tennessee Titans 16 476 2187 16  75 4.6 136.7  16  -4.29 1.0000
## 4   4  San Francisco 49ers 16 458 2019 15  47 4.4 126.2  29 -16.25 0.9375
## 5   5      Atlanta Falcons 16 421 1928 20  75 4.6 120.5   8  20.78 1.2500
## 6   6      Oakland Raiders 16 434 1922 17  75 4.4 120.1  17 -38.09 1.0625
## 7   7 New England Patriots 16 482 1872 19  44 3.9 117.0  27 -36.10 1.1875
## 8   8       Houston Texans 16 456 1859  8  45 4.1 116.2  20 -39.31 0.5000
## 9   9       Miami Dolphins 16 406 1824 14  62 4.5 114.0  26 -29.06 0.8750
## 10 10    Carolina Panthers 16 453 1814 16  47 4.0 113.4  13 -24.98 1.0000
## 11 11  Philadelphia Eagles 16 438 1813 16  30 4.1 113.3  24 -17.58 1.0000
## 12 12        New York Jets 16 418 1802 10  35 4.3 112.6  24  -8.50 0.6250
## 13 13   Cincinnati Bengals 16 446 1769 17  74 4.0 110.6  20   1.28 1.0625
## 14 14  Pittsburgh Steelers 16 409 1760 13  60 4.3 110.0  15 -16.84 0.8125
## 15 15   Kansas City Chiefs 16 412 1748 15  70 4.2 109.3  18 -24.46 0.9375
## 16 16   New Orleans Saints 16 404 1742 17  75 4.3 108.9  17  15.68 1.0625
## 17 17        Chicago Bears 16 380 1735 10  69 4.6 108.4  26 -19.89 0.6250
## 18 18    Arizona Cardinals 16 399 1732 20  58 4.3 108.3  27 -14.69 1.2500
## 19 19     Cleveland Browns 16 350 1712 13  85 4.9 107.0  26 -15.44 0.8125
## 20 20    Green Bay Packers 16 374 1701 11  61 4.5 106.3  21  -3.65 0.6875
## 21 21  Washington Redskins 16 379 1696 17  66 4.5 106.0  19  -6.19 1.0625
## 22 22 Jacksonville Jaguars 16 392 1631  8  57 4.2 101.9  23 -44.63 0.5000
## 23 23   Indianapolis Colts 16 409 1628 13  33 4.0 101.8  16  -4.97 0.8125
## 24 24 Tampa Bay Buccaneers 16 453 1616  8  45 3.6 101.0  19 -68.53 0.5000
## 25 25     Seattle Seahawks 16 403 1591 13  75 3.9  99.4  22 -33.05 0.8125
## 26 26   San Diego Chargers 16 398 1510 10  48 3.8  94.4  22 -32.24 0.6250
## 27 27       Denver Broncos 16 410 1484 11  64 3.6  92.8  24 -57.61 0.6875
## 28 28    Baltimore Ravens 16 367 1463 10  41 4.0  91.4  19 -11.25 0.6250
## 29 29      New York Giants 16 398 1412  6  25 3.5  88.3  22 -60.21 0.3750
## 30 30        Detroit Lions 16 350 1310  9  42 3.7  81.9   9 -17.32 0.5625
## 31 31     Los Angeles Rams 16 375 1252  9  30 3.3  78.3  24 -54.51 0.5625
## 32 32    Minnesota Vikings 16 380 1205  9  36 3.2  75.3  20 -78.99 0.5625
## 33 NA                     NA  NA   NA NA  NA  NA    NA  NA     NA     NA
## 34 NA                     NA  NA   NA NA  NA  NA    NA  NA     NA     NA
## 35 NA                     NA  NA   NA NA  NA  NA    NA  NA     NA     NA
```

Recall that if we want to save these variables instead of just viewing them, we need to either assign them to something.

```
team_rushing <- mutate(team_rushing, TD.G = TD / G)
team_rushing
```

```
##    Rk                  Tm  G Att  Yds TD Lng Y.A   Y.G Fmb    EXP   TD.G
## 1   1        Buffalo Bills 16 492 2630 29  75 5.3 164.4  15  59.91 1.8125
## 2   2       Dallas Cowboys 16 499 2396 24  60 4.8 149.8  22  39.96 1.5000
## 3   3     Tennessee Titans 16 476 2187 16  75 4.6 136.7  16  -4.29 1.0000
## 4   4  San Francisco 49ers 16 458 2019 15  47 4.4 126.2  29 -16.25 0.9375
## 5   5      Atlanta Falcons 16 421 1928 20  75 4.6 120.5   8  20.78 1.2500
## 6   6      Oakland Raiders 16 434 1922 17  75 4.4 120.1  17 -38.09 1.0625
## 7   7 New England Patriots 16 482 1872 19  44 3.9 117.0  27 -36.10 1.1875
## 8   8       Houston Texans 16 456 1859  8  45 4.1 116.2  20 -39.31 0.5000
## 9   9       Miami Dolphins 16 406 1824 14  62 4.5 114.0  26 -29.06 0.8750
## 10 10    Carolina Panthers 16 453 1814 16  47 4.0 113.4  13 -24.98 1.0000
## 11 11  Philadelphia Eagles 16 438 1813 16  30 4.1 113.3  24 -17.58 1.0000
## 12 12        New York Jets 16 418 1802 10  35 4.3 112.6  24  -8.50 0.6250
## 13 13   Cincinnati Bengals 16 446 1769 17  74 4.0 110.6  20   1.28 1.0625
## 14 14  Pittsburgh Steelers 16 409 1760 13  60 4.3 110.0  15 -16.84 0.8125
## 15 15   Kansas City Chiefs 16 412 1748 15  70 4.2 109.3  18 -24.46 0.9375
## 16 16   New Orleans Saints 16 404 1742 17  75 4.3 108.9  17  15.68 1.0625
## 17 17        Chicago Bears 16 380 1735 10  69 4.6 108.4  26 -19.89 0.6250
## 18 18    Arizona Cardinals 16 399 1732 20  58 4.3 108.3  27 -14.69 1.2500
## 19 19     Cleveland Browns 16 350 1712 13  85 4.9 107.0  26 -15.44 0.8125
## 20 20    Green Bay Packers 16 374 1701 11  61 4.5 106.3  21  -3.65 0.6875
## 21 21  Washington Redskins 16 379 1696 17  66 4.5 106.0  19  -6.19 1.0625
## 22 22 Jacksonville Jaguars 16 392 1631  8  57 4.2 101.9  23 -44.63 0.5000
## 23 23   Indianapolis Colts 16 409 1628 13  33 4.0 101.8  16  -4.97 0.8125
## 24 24 Tampa Bay Buccaneers 16 453 1616  8  45 3.6 101.0  19 -68.53 0.5000
## 25 25     Seattle Seahawks 16 403 1591 13  75 3.9  99.4  22 -33.05 0.8125
## 26 26   San Diego Chargers 16 398 1510 10  48 3.8  94.4  22 -32.24 0.6250
## 27 27       Denver Broncos 16 410 1484 11  64 3.6  92.8  24 -57.61 0.6875
## 28 28     Baltimore Ravens 16 367 1463 10  41 4.0  91.4  19 -11.25 0.6250
## 29 29      New York Giants 16 398 1412  6  25 3.5  88.3  22 -60.21 0.3750
## 30 30        Detroit Lions 16 350 1310  9  42 3.7  81.9   9 -17.32 0.5625
## 31 31     Los Angeles Rams 16 375 1252  9  30 3.3  78.3  24 -54.51 0.5625
## 32 32    Minnesota Vikings 16 380 1205  9  36 3.2  75.3  20 -78.99 0.5625
## 33 NA                      NA  NA   NA NA  NA  NA    NA  NA     NA     NA
## 34 NA                      NA  NA   NA NA  NA  NA    NA  NA     NA     NA
## 35 NA                      NA  NA   NA NA  NA  NA    NA  NA     NA     NA
```

Creation Functions

Mutating allows us to create new variables but the arguments that we pass in to create the new variable must be vectors that return an equivalent number of values as the output. These functions are just some of the possible tools use can use to create new variables:

Arithmetic Operators: '+, -, *, /, ^'

Modular Arithmetic: '%/%' Integer Division, '%%' Remainder

Cumulative and Rolling Aggregates: 'cumsum()' Cumulative Sums, 'cumprod()' Cumulative products, 'cummean()' Cumulative Means

Offsets: 'lead()' Leading Values, 'lag()' Lagging Values

These will all be useful in different ways that allow you to play with existing variables in order to create new variables.

Summaries

The summary function collapses a data frame into a single row. However, we first need to discuss the 'group_by()' function. It is only when it is used in conjunction with 'summary()' is when the function can be used to its fullest. The group by function allows us to create a grouped data frame in which dplyr verbs will be automatically applied by group instead of to individual observations. Our team_rushing data frame is a grouped data set in which players are grouped by teams. Normally you would have a dataset of every single player in the NFl and their rushing statistics, however, we grouped the data by Team so that we calculate the team rushing statistics instead of player rushing statistics. This enables us to see which teams are the best at running the football as opposed to individual players in the NFL for the 2016 season. Now we can use summarize on the grouped data frame. Normally you would use group_by() in the argument of summarize but in this case, since team_rushing is already a grouped data frame, that is not necessary.

```
summarize(team_rushing, mean(TD, na.rm = TRUE)) #average number of touchdowns for every team in the NFL for the 20
16                                                NFL season
```

```
##   mean(TD, na.rm = TRUE)
## 1               13.84375
```

Conclusion

In conclusion, the *dplyr* package is an extremely useful tool that can be used in order to manipulate data in order to better suit the needs of the user. Through the package we can reorder and filter data frames, manipulate them to add new columns, and collapse the entire frame to return a summary of the dataset. Using *dplyr*, we are able to manipulate data from all sorts of origins, from science to society.

References

1. NFL_team_rushing_offense from profootballreference.com
2. R for Data Science by Hadley Wickham & Garrett Grolemund
3. https://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf

4. https://www.youtube.com/watch?v=Knko9hkL3q0
5. https://github.com/ucb-stat133/stat133-fall-2017/blob/master/tutorials/05-intro-to-dplyr.md
6. http://r4ds.had.co.nz/diagrams/transform-logical.png
7. https://www.rdocumentation.org/packages/Hmisc/versions/4.0-3/topics/summarize
8. https://www.r-bloggers.com/using-mutate-from-dplyr-inside-a-function-getting-around-non-standard-evaluation/