Data Manipulation with R

Armand Tossou

9/5/2021

Guide to oUsing Dplyr

We'll install `dplyr`, call the library and explore the following built-in functions: - filter() and slice() - arrange() - select() and rename() - distinct() - mutate() and transmute() - summarise() - sample_n() and sample_frac()

Install and call the dplyr library

```
# install dplyr package
#install.packages("dplyr")

library(dplyr)
```

```
##
## 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
```

```
# install some data packages
## a large data set of flights from 2013
#install.packages("nycflights13")
library(nycflights13)

# call the flights data
head(flights)
```

```
## # A tibble: 6 x 19
##    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
## 1  2013     1     1      517            515         2      830            819
## 2  2013     1     1      533            529         4      850            830
## 3  2013     1     1      542            540         2      923            850
```

```
## 4  2013     1     1      544            545        -1     1004           1022
## 5  2013     1     1      554            600        -6      812            837
## 6  2013     1     1      554            558        -4      740            728
## # ... with 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

```
summary(flights)
```

```
##       year          month             day           dep_time    sched_dep_time
##  Min.   :2013   Min.   : 1.000   Min.   : 1.00   Min.   :   1   Min.   : 106
##  1st Qu.:2013   1st Qu.: 4.000   1st Qu.: 8.00   1st Qu.: 907   1st Qu.: 906
##  Median :2013   Median : 7.000   Median :16.00   Median :1401   Median :1359
##  Mean   :2013   Mean   : 6.549   Mean   :15.71   Mean   :1349   Mean   :1344
##  3rd Qu.:2013   3rd Qu.:10.000   3rd Qu.:23.00   3rd Qu.:1744   3rd Qu.:1729
##  Max.   :2013   Max.   :12.000   Max.   :31.00   Max.   :2400   Max.   :2359
##                                                  NA's   :8255
##    dep_delay          arr_time    sched_arr_time   arr_delay
##  Min.   : -43.00   Min.   :   1   Min.   :   1   Min.   : -86.000
##  1st Qu.:  -5.00   1st Qu.:1104   1st Qu.:1124   1st Qu.: -17.000
##  Median :  -2.00   Median :1535   Median :1556   Median :  -5.000
##  Mean   :  12.64   Mean   :1502   Mean   :1536   Mean   :   6.895
##  3rd Qu.:  11.00   3rd Qu.:1940   3rd Qu.:1945   3rd Qu.:  14.000
##  Max.   :1301.00   Max.   :2400   Max.   :2359   Max.   :1272.000
##  NA's   :8255      NA's   :8713                  NA's   :9430
##    carrier              flight       tailnum             origin
##  Length:336776      Min.   :   1   Length:336776      Length:336776
##  Class :character   1st Qu.: 553   Class :character   Class :character
##  Mode  :character   Median :1496   Mode  :character   Mode  :character
##                     Mean   :1972
##                     3rd Qu.:3465
##                     Max.   :8500
##
##      dest              air_time        distance         hour
##  Length:336776      Min.   : 20.0   Min.   :  17   Min.   : 1.00
##  Class :character   1st Qu.: 82.0   1st Qu.: 502   1st Qu.: 9.00
##  Mode  :character   Median :129.0   Median : 872   Median :13.00
##                     Mean   :150.7   Mean   :1040   Mean   :13.18
##                     3rd Qu.:192.0   3rd Qu.:1389   3rd Qu.:17.00
##                     Max.   :695.0   Max.   :4983   Max.   :23.00
##                     NA's   :9430
##      minute        time_hour
##  Min.   : 0.00   Min.   :2013-01-01 05:00:00
##  1st Qu.: 8.00   1st Qu.:2013-04-04 13:00:00
##  Median :29.00   Median :2013-07-03 10:00:00
##  Mean   :26.23   Mean   :2013-07-03 05:22:54
##  3rd Qu.:44.00   3rd Qu.:2013-10-01 07:00:00
##  Max.   :59.00   Max.   :2013-12-31 23:00:00
##
```

Explore the filter() function

`filter()` allows us to select as subset of rows in a dataframe.

```
# flights from American Airlines that occured on November 3rd.
head(filter(flights, month==11, day==3, carrier=='AA'))
```

```
## # A tibble: 6 x 19
##    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
## 1  2013    11     3      538            545        -7      824            855
## 2  2013    11     3      556            600        -4      900            905
## 3  2013    11     3      604            610        -6      844            855
## 4  2013    11     3      624            629        -5      907            929
## 5  2013    11     3      625            630        -5      736            805
## 6  2013    11     3      653            655        -2      925            920
## # ... with 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

```
# trying bracket notation to achieve the same filter result
head(flights[flights$month==11 & flights$day==3 & flights$carrier=='AA',])
```

```
## # A tibble: 6 x 19
##    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
## 1  2013    11     3      538            545        -7      824            855
## 2  2013    11     3      556            600        -4      900            905
## 3  2013    11     3      604            610        -6      844            855
## 4  2013    11     3      624            629        -5      907            929
## 5  2013    11     3      625            630        -5      736            805
## 6  2013    11     3      653            655        -2      925            920
## # ... with 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

Explore the slice() function

`slice()` allows us to select rows by position.

```
# first 10 rows
slice(flights, 1:10)
```

```
## # A tibble: 10 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     1     1      517            515         2      830            819
##  2  2013     1     1      533            529         4      850            830
##  3  2013     1     1      542            540         2      923            850
##  4  2013     1     1      544            545        -1     1004           1022
##  5  2013     1     1      554            600        -6      812            837
##  6  2013     1     1      554            558        -4      740            728
##  7  2013     1     1      555            600        -5      913            854
##  8  2013     1     1      557            600        -3      709            723
##  9  2013     1     1      557            600        -3      838            846
```

```
## 10  2013     1     1      558            600        -2      753            745
## # ... with 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

Explore the arrange() function

`arrange()` works similarly to filter(), except it re-orders the rows instead of selecting/filtering them.

```
# first 6 rows, ordered by some key columns
head(arrange(flights,year,month,day,arr_time))
```

```
## # A tibble: 6 x 19
##    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
## 1  2013     1     1     1929           1920         9        3              7
## 2  2013     1     1     2121           2040        41        6           2323
## 3  2013     1     1     2058           2100        -2        8           2359
## 4  2013     1     1     2120           2130       -10       16             18
## 5  2013     1     1     2134           2045        49       20           2352
## 6  2013     1     1     2312           2000       192       21           2110
## # ... with 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

```
# first 6 rows, ordered by some key columns
## do descending order for arrival time
head(arrange(flights,year,month,day,desc(arr_time)))
```

```
## # A tibble: 6 x 19
##    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
## 1  2013     1     1     2209           2155        14     2400           2337
## 2  2013     1     1     1952           1930        22     2358           2207
## 3  2013     1     1     2025           2028        -3     2358           2351
## 4  2013     1     1     2119           1930       109     2358           2136
## 5  2013     1     1     2052           2045         7     2357           2359
## 6  2013     1     1     2030           2035        -5     2354           2342
## # ... with 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

Explore the select() function

`select()` allows us to zoom in on a few columns of interest.

```
head(select(flights, carrier, arr_time, month))
```

```
## # A tibble: 6 x 3
##   carrier arr_time month
##   <chr>      <int> <int>
```

```
## 1 UA           830     1
## 2 UA           850     1
## 3 AA           923     1
## 4 B6          1004     1
## 5 DL           812     1
## 6 UA           740     1
```

Explore the rename() function

`rename()` allows us to quickly rename column names.

```
# let's rename the 'carrier' column as 'airline_carrier'
head(rename(flights, airline_carrier = carrier))
```

```
## # A tibble: 6 x 19
##    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
## 1  2013     1     1      517            515         2      830            819
## 2  2013     1     1      533            529         4      850            830
## 3  2013     1     1      542            540         2      923            850
## 4  2013     1     1      544            545        -1     1004           1022
## 5  2013     1     1      554            600        -6      812            837
## 6  2013     1     1      554            558        -4      740            728
## # ... with 11 more variables: arr_delay <dbl>, airline_carrier <chr>,
## #   flight <int>, tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>,
## #   distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

Explore the distinct() function

`distinct()` allows us to select the unique values in a column.

```
# find out the list of unique aireline carriers in the flights dataset
distinct(select(flights,carrier))
```

```
## # A tibble: 16 x 1
##    carrier
##    <chr>
##  1 UA
##  2 AA
##  3 B6
##  4 DL
##  5 EV
##  6 MQ
##  7 US
##  8 WN
##  9 VX
## 10 FL
## 11 AS
## 12 9E
## 13 F9
## 14 HA
## 15 YV
## 16 OO
```

Explore the mutate() function

`mutate()` allows us to add new columns that are functions of existing columns.

```
# create a new column that is the difference between arrival delay and departure delay
mutate(flights,new_col = arr_delay - dep_delay)
```

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

Explore the transmute() function

`transmute()` is related to `mutate()`. We use it if we just want the newly created column back.

```
# create a new column that is the difference between arrival delay and departure delay
transmute(flights,new_col = arr_delay - dep_delay)
```

```
## # A tibble: 336,776 x 1
##    new_col
##      <dbl>
##  1       9
##  2      16
##  3      31
##  4     -17
##  5     -19
##  6      16
##  7      24
##  8     -11
##  9      -5
## 10      10
## # ... with 336,766 more rows
```

Explore the summarise() function

`summarise()` allows us to collapse the dataframe into single rows using some sort of a function that aggregate results.

```
# compute average air time, with missing values removed
summarise(flights, avg_air_time=mean(air_time,na.rm=TRUE))
```

```
## # A tibble: 1 x 1
##   avg_air_time
##          <dbl>
## 1         151.
```

```
# compute total airtime for all the flights
summarise(flights,total_time=sum(air_time,na.rm=TRUE))
```

```
## # A tibble: 1 x 1
##   total_time
##        <dbl>
## 1   49326610
```

Explore the sample_n() function

`sample_n()` allows us to take random samples of the data frame, by specifying a number to pick.

```
# take a random sample of 10 rows from the flights data frame
sample_n(flights,10)
```

```
## # A tibble: 10 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
##  1  2013     6    17      803            759         4     1017           1026
##  2  2013     5    10     1142           1145        -3     1307           1322
##  3  2013     6     7     2129           2117        12     2348           2310
##  4  2013     7    11     2012           1940        32     2221           2142
##  5  2013     1    14     1808           1820       -12     2005           2005
##  6  2013     1     1     1059           1100        -1     1210           1215
##  7  2013     4    12      600            601        -1      919            933
##  8  2013     9     3     1910           1915        -5     2207           2223
##  9  2013     3     2     1919           1910         9     2037           2035
## 10  2013     6    29      118           2359        79      445            340
## # ... with 11 more variables: arr_delay <dbl>, carrier <chr>, flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
## #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

Explore the sample_frac() function

`sample_frac()` allows us to take random samples of the data frame, by specifying a percentage to pick.

```
# take a random sample of 10% of the rows from the flights data frame
sample_frac(flights,0.1)
```

```
## # A tibble: 33,678 x 19
##     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
```

```
##  1  2013     5    12      723           700        23      959           1019
##  2  2013    12    26     2218          2058        80       21           2313
##  3  2013     9    26      657           655         2      925            920
##  4  2013     4    20     1144          1144         0     1405           1358
##  5  2013    11    18     2058          2105        -7     2340             11
##  6  2013     7    22     1202          1145        17     1405           1340
##  7  2013     5     3     1658          1650         8     1825           1845
##  8  2013     9    28     1601          1600         1     1931           1905
##  9  2013     9    11     1100          1110       -10     1240           1311
## 10  2013     8    14     2058          2100        -2       36             12
## # ... with 33,668 more rows, and 11 more variables: arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

Pipe operator

The pipe operator `%>%` is really handy while working with `dplyr` libraries. It allows us to chain multiple operations together.

```
#load the built-in mtcars data frame
df <- mtcars

# motivation for the pipe %>% operator:
## nesting: makes interpretation of the code hard
result <- arrange(sample_n(filter(df,mpg>20), size=5),desc(mpg))

print(result)
```

```
##                mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Fiat 128      32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## Fiat X1-9     27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Porsche 914-2 26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## Toyota Corona 21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## Mazda RX4     21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
```

```
# using a multiple assignments approach instead
a <- filter(df,mpg>20)
b <- sample_n(a, size = 5)
result <- arrange(b,desc(mpg))

print(result)
```

```
##                 mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Toyota Corolla 33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## Fiat 128       32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## Merc 240D      24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Volvo 142E     21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
## Mazda RX4 Wag  21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
```

```
# re-write all of this using the pipe operator: %>%
## syntax: Data %>% operation1 %>% operation2 %>% operation3 ...
results <- df %>% filter(mpg>20) %>% sample_n(size = 5) %>% arrange(desc(mpg))
print(result)
```

```
##                 mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Toyota Corolla 33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## Fiat 128       32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## Merc 240D      24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Volvo 142E     21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
## Mazda RX4 Wag  21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
```

Dplyr Exercises

We will use the mtcars dataframe for this exercise!

```
head(mtcars)
```

```
##                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

EX1. Return rows of cars that have an mpg value greater than 20 and 6 cylinders.

```
filter(mtcars,mpg>20,cyl==6)
```

```
##                 mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4      21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag  21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Hornet 4 Drive 21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
```

Ex2. Reorder the Data Frame by cyl first, then by descending wt.

```
arrange(mtcars,cyl,desc(wt))
```

```
##                    mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Merc 240D          24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Merc 230           22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Volvo 142E         21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
## Toyota Corona      21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## Datsun 710         22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Fiat 128           32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
```

```
## Porsche 914-2       26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## Fiat X1-9           27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Toyota Corolla      33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## Honda Civic         30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## Lotus Europa        30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Valiant             18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Merc 280            19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## Merc 280C           17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
## Hornet 4 Drive      21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Mazda RX4 Wag       21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Ferrari Dino        19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
## Mazda RX4           21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
## Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
## Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
## Merc 450SE          16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
## Pontiac Firebird    19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
## Camaro Z28          13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
## Merc 450SLC         15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
## Merc 450SL          17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
## Duster 360          14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## Maserati Bora       15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
## Dodge Challenger    15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
## Hornet Sportabout   18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## AMC Javelin         15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
## Ford Pantera L      15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
```

Ex3. Select the columns mpg and hp.

```
select(mtcars,mpg,hp)
```

```
##                      mpg  hp
## Mazda RX4           21.0 110
## Mazda RX4 Wag       21.0 110
## Datsun 710          22.8  93
## Hornet 4 Drive      21.4 110
## Hornet Sportabout   18.7 175
## Valiant             18.1 105
## Duster 360          14.3 245
## Merc 240D           24.4  62
## Merc 230            22.8  95
## Merc 280            19.2 123
## Merc 280C           17.8 123
## Merc 450SE          16.4 180
## Merc 450SL          17.3 180
## Merc 450SLC         15.2 180
## Cadillac Fleetwood  10.4 205
## Lincoln Continental 10.4 215
## Chrysler Imperial   14.7 230
## Fiat 128            32.4  66
## Honda Civic         30.4  52
## Toyota Corolla      33.9  65
```

```
## Toyota Corona       21.5  97
## Dodge Challenger    15.5 150
## AMC Javelin         15.2 150
## Camaro Z28          13.3 245
## Pontiac Firebird    19.2 175
## Fiat X1-9           27.3  66
## Porsche 914-2       26.0  91
## Lotus Europa        30.4 113
## Ford Pantera L      15.8 264
## Ferrari Dino        19.7 175
## Maserati Bora       15.0 335
## Volvo 142E          21.4 109
```

Ex4. Select the distinct values of the gear column.

```
distinct(mtcars,gear)
```

```
##                gear
## Mazda RX4         4
## Hornet 4 Drive    3
## Porsche 914-2     5
```

Ex5. Create a new column called "Performance" which is calculated by hp divided by wt.

```
mutate(mtcars,performance = hp / wt)
```

```
##                     mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4           21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag       21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710          22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive      21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout   18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## Valiant             18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Duster 360          14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## Merc 240D           24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Merc 230            22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Merc 280            19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## Merc 280C           17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
## Merc 450SE          16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
## Merc 450SL          17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
## Merc 450SLC         15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
## Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
## Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
## Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
## Fiat 128            32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## Honda Civic         30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## Toyota Corolla      33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## Toyota Corona       21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## Dodge Challenger    15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
```

```
## AMC Javelin         15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
## Camaro Z28          13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
## Pontiac Firebird    19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
## Fiat X1-9           27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Porsche 914-2       26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## Lotus Europa        30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Ford Pantera L      15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
## Ferrari Dino        19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
## Maserati Bora       15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
## Volvo 142E          21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
##                     performance
## Mazda RX4             41.98473
## Mazda RX4 Wag         38.26087
## Datsun 710            40.08621
## Hornet 4 Drive        34.21462
## Hornet Sportabout     50.87209
## Valiant               30.34682
## Duster 360            68.62745
## Merc 240D             19.43574
## Merc 230              30.15873
## Merc 280              35.75581
## Merc 280C             35.75581
## Merc 450SE            44.22604
## Merc 450SL            48.25737
## Merc 450SLC           47.61905
## Cadillac Fleetwood    39.04762
## Lincoln Continental   39.63864
## Chrysler Imperial     43.03087
## Fiat 128              30.00000
## Honda Civic           32.19814
## Toyota Corolla        35.42234
## Toyota Corona         39.35091
## Dodge Challenger      42.61364
## AMC Javelin           43.66812
## Camaro Z28            63.80208
## Pontiac Firebird      45.51365
## Fiat X1-9             34.10853
## Porsche 914-2         42.52336
## Lotus Europa          74.68605
## Ford Pantera L        83.28076
## Ferrari Dino          63.17690
## Maserati Bora         93.83754
## Volvo 142E            39.20863
```

Ex6. Find the mean mpg value using dplyr.

```
summarise(mtcars,avg_mpg=mean(mpg))
```

```
##    avg_mpg
## 1 20.09062
```

Ex7. Use pipe operators to get the mean hp value for cars with 6 cylinders.

```
mtcars %>% filter(cyl==6) %>% summarise(avg_hp=mean(hp,na.rm=TRUE))
```

```
##     avg_hp
## 1 122.2857
```

Guide to using Tidyr

- Install and load packages
- Explore the four most important functions in the `tidyr` library:
  - `gather`
  - `spread`
  - `separate`
  - and `unite`
- Explore the `data.table` package. A `data.table` is essentially a `data.frame` with a few more features. This offers the advantages of speed and a cleaner syntax.

```
# install the tidyr package
#install.packages("tidyr")
library(tidyr)

# install a complementary data package

#install.packages("data.table")
library(data.table)
```

```
##
## Attaching package: 'data.table'

## The following objects are masked from 'package:dplyr':
##
##     between, first, last
```

Explore the gather function

Collapse multiple columns into key-pair values.

```
# let's create a data frame of quarterly returns for a company
comp  <- c(1,1,1,2,2,2,3,3,3)
yr <- c(1998:2000)
q1 <- runif(9, min = 0, max = 100)
q2 <- runif(9, min = 0, max = 100)
q3 <- runif(9, min = 0, max = 100)
q4 <- runif(9, min = 0, max = 100)

df <- data.frame(comp=comp,year=yr,Qtr1=q1,Qtr2=q2,Qtr3=q3,Qtr4=q4)
df
```

```
##   comp year      Qtr1      Qtr2      Qtr3     Qtr4
## 1    1 1998 33.540065 89.224749 80.984158 17.08066
## 2    1 1999 18.821550 16.115924 85.108758 98.34004
## 3    1 2000 63.117048  4.580671 76.743418 43.23469
## 4    2 1998 65.717532 92.811854 28.871073 84.34252
## 5    2 1999  6.069785 60.067674 16.039696 37.89654
## 6    2 2000  3.216528 99.105778 59.906984 81.53807
## 7    3 1998 92.321535 54.032708 62.513946 22.00980
## 8    3 1999  7.936434 50.697767  9.953596 91.00707
## 9    3 2000 68.701970 80.944598 62.331147 30.67073
```

This data frame is in wide format.

```
# Use the gather() function
# to restructure the time component of the data into long format
df_long <- gather(df,Quarter,Revenue,Qtr1:Qtr4)
df_long
```

```
##    comp year Quarter   Revenue
## 1     1 1998    Qtr1 33.540065
## 2     1 1999    Qtr1 18.821550
## 3     1 2000    Qtr1 63.117048
## 4     2 1998    Qtr1 65.717532
## 5     2 1999    Qtr1  6.069785
## 6     2 2000    Qtr1  3.216528
## 7     3 1998    Qtr1 92.321535
## 8     3 1999    Qtr1  7.936434
## 9     3 2000    Qtr1 68.701970
## 10    1 1998    Qtr2 89.224749
## 11    1 1999    Qtr2 16.115924
## 12    1 2000    Qtr2  4.580671
## 13    2 1998    Qtr2 92.811854
## 14    2 1999    Qtr2 60.067674
## 15    2 2000    Qtr2 99.105778
## 16    3 1998    Qtr2 54.032708
## 17    3 1999    Qtr2 50.697767
## 18    3 2000    Qtr2 80.944598
## 19    1 1998    Qtr3 80.984158
## 20    1 1999    Qtr3 85.108758
## 21    1 2000    Qtr3 76.743418
## 22    2 1998    Qtr3 28.871073
## 23    2 1999    Qtr3 16.039696
## 24    2 2000    Qtr3 59.906984
## 25    3 1998    Qtr3 62.513946
## 26    3 1999    Qtr3  9.953596
## 27    3 2000    Qtr3 62.331147
## 28    1 1998    Qtr4 17.080661
## 29    1 1999    Qtr4 98.340044
## 30    1 2000    Qtr4 43.234686
## 31    2 1998    Qtr4 84.342517
## 32    2 1999    Qtr4 37.896540
## 33    2 2000    Qtr4 81.538070
## 34    3 1998    Qtr4 22.009804
```

```
## 35    3 1999    Qtr4 91.007071
## 36    3 2000    Qtr4 30.670730
```

Explore the spread function

`spread()` is complementary to `gather()`.

```
# create a stocks dataset
stocks <- data.frame(
  time = as.Date('2009-01-01') + 0:9,
  x = rnorm(10, 0, 1),
  y = rnorm(10, 0, 2),
  z = rnorm(10, 0, 4)
)

stocks
```

```
##          time           x          y          z
## 1  2009-01-01  1.19003455 -1.0846408 -0.1609892
## 2  2009-01-02  0.05111443 -2.1449610 -2.0166535
## 3  2009-01-03 -0.55713314 -1.4116419 -5.8126093
## 4  2009-01-04 -1.63254264  0.3940419 -0.6577095
## 5  2009-01-05  2.10603242  3.0111423 -4.1626933
## 6  2009-01-06  0.67380936 -0.1755575  6.3826478
## 7  2009-01-07 -0.09360460  1.5590123  0.5115624
## 8  2009-01-08 -0.71142503 -2.9019360  5.8505307
## 9  2009-01-09  1.46165902 -1.6852852  2.3557081
## 10 2009-01-10  0.14581562  1.8570893 -4.7355260
```

```
# gather the data
stocks.gathered <- stocks %>% gather(stock,price,x,y,z)
head(stocks.gathered)
```

```
##         time stock       price
## 1 2009-01-01     x  1.19003455
## 2 2009-01-02     x  0.05111443
## 3 2009-01-03     x -0.55713314
## 4 2009-01-04     x -1.63254264
## 5 2009-01-05     x  2.10603242
## 6 2009-01-06     x  0.67380936
```

```
# let's spread this dataset back out
## this will generate a data frame in wide format
stocks.gathered %>% spread(stock,price)
```

```
##          time           x          y          z
## 1  2009-01-01  1.19003455 -1.0846408 -0.1609892
## 2  2009-01-02  0.05111443 -2.1449610 -2.0166535
## 3  2009-01-03 -0.55713314 -1.4116419 -5.8126093
## 4  2009-01-04 -1.63254264  0.3940419 -0.6577095
## 5  2009-01-05  2.10603242  3.0111423 -4.1626933
```

```
## 6  2009-01-06  0.67380936 -0.1755575  6.3826478
## 7  2009-01-07 -0.09360460  1.5590123  0.5115624
## 8  2009-01-08 -0.71142503 -2.9019360  5.8505307
## 9  2009-01-09  1.46165902 -1.6852852  2.3557081
## 10 2009-01-10  0.14581562  1.8570893 -4.7355260
```

```
# another way to spread the data
spread(stocks.gathered,time,price)
```

```
##   stock 2009-01-01  2009-01-02 2009-01-03 2009-01-04 2009-01-05 2009-01-06
## 1     x  1.1900345  0.05111443 -0.5571331 -1.6325426   2.106032  0.6738094
## 2     y -1.0846408 -2.14496104 -1.4116419  0.3940419   3.011142 -0.1755575
## 3     z -0.1609892 -2.01665345 -5.8126093 -0.6577095  -4.162693  6.3826478
##   2009-01-07 2009-01-08 2009-01-09 2009-01-10
## 1 -0.0936046  -0.711425   1.461659  0.1458156
## 2  1.5590123  -2.901936  -1.685285  1.8570893
## 3  0.5115624   5.850531   2.355708 -4.7355260
```

Explore the separate function

`separate` allows us to convert a single character column into multiple columns, using regular expressions or vector / character expressions.

```
# let's generate a dataframe
df <- data.frame(new.col=c(NA,"a.x","b.y","c.z"))
df
```

```
##   new.col
## 1    <NA>
## 2     a.x
## 3     b.y
## 4     c.z
```

```
separate(df,new.col,c("ABC","XYZ"))
```

```
##    ABC  XYZ
## 1 <NA> <NA>
## 2    a    x
## 3    b    y
## 4    c    z
```

The `separate()` function uses a non-alphanumeric character to split the column name. Now let's specify the separator.

```
# let's generate a dataframe
df <- data.frame(new.col=c(NA,"a-x","b-y","c-z"))
df
```

```
##   new.col
## 1    <NA>
## 2     a-x
## 3     b-y
## 4     c-z
```

```
df.sep <- separate(data=df,col=new.col,into=c("ABC","XYZ"),sep = "-")
df.sep
```

```
##    ABC  XYZ
## 1 <NA> <NA>
## 2    a    x
## 3    b    y
## 4    c    z
```

Explore the unite function

`unite()` is the exact opposite of `separate()`.

```
unite(data=df.sep,col=new.joined.col,ABC,XYZ,sep = "---")
```

```
##   new.joined.col
## 1        NA---NA
## 2          a---x
## 3          b---y
## 4          c---z
```