Dplyr Introduction

Matthew Flickinger

July 12, 2017

Introduction to Dplyr

This document gives an overview of many of the features of the dplyr library include in the "tidyverse" of related R pacakges. First we will load the library and a sample dataset.

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

```
#install.packages("nycflights13")
library(nycflights13)
# Show fewer rows by default in this document
options(tibble.print_min = 5L, tibble.print_max = 5L)
```

We will primarly be using the `flights` data

```
flights
```

```
## # A tibble: 336,776 x 19
##    year month   day dep_time sched_dep_time dep_delay arr_time
##   <int> <int> <int>    <int>          <int>     <dbl>    <int>
## 1  2013     1     1      517            515         2      830
## 2  2013     1     1      533            529         4      850
## 3  2013     1     1      542            540         2      923
## 4  2013     1     1      544            545        -1     1004
## 5  2013     1     1      554            600        -6      812
## # ... with 336,771 more rows, and 12 more variables: sched_arr_time <int>,
## #   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>
```

Filtering Rows

Find all flights from Detroit in June (in 2013)

```
# Same as using base R
# flights[flights$dest=="DTW" & flights$month==6, ]
# subset(flights, dest=="DTW" & month==6)
filter(flights, dest=="DTW" & month==6)
```

```
## # A tibble: 807 x 19
##    year month   day dep_time sched_dep_time dep_delay arr_time
##   <int> <int> <int>    <int>          <int>     <dbl>    <int>
## 1  2013     6     1      559            600        -1      739
## 2  2013     6     1      709            715        -6      846
## 3  2013     6     1      748            756        -8      929
## 4  2013     6     1      758            755         3      951
## 5  2013     6     1      832            829         3     1017
## # ... with 802 more rows, and 12 more variables: sched_arr_time <int>,
## #   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>
```

`filter()` expects a data source as the first parameter, and a single expression as the second parameter. Combine multiple criteria with `&` for "and" – `|` for "or".

Selecting Columns

```
# List columns
select(flights, dep_time, arr_time, carrier)
```

```
## # A tibble: 336,776 x 3
##   dep_time arr_time carrier
##      <int>    <int>   <chr>
## 1      517      830      UA
## 2      533      850      UA
## 3      542      923      AA
## 4      544     1004      B6
## 5      554      812      DL
## # ... with 336,771 more rows
```

```
# Exclude columns
select(flights, -year, -tailnum)
```

```
## # A tibble: 336,776 x 17
##   month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
##   <int> <int>    <int>          <int>     <dbl>    <int>          <int>
## 1     1     1      517            515         2      830            819
## 2     1     1      533            529         4      850            830
## 3     1     1      542            540         2      923            850
## 4     1     1      544            545        -1     1004           1022
## 5     1     1      554            600        -6      812            837
## # ... with 336,771 more rows, and 10 more variables: arr_delay <dbl>,
## #   carrier <chr>, flight <int>, origin <chr>, dest <chr>, air_time <dbl>,
## #   distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

```
# Select column range (in data.frame order)
select(flights, month:dep_delay)
```

```
## # A tibble: 336,776 x 5
##   month   day dep_time sched_dep_time dep_delay
##   <int> <int>    <int>          <int>     <dbl>
## 1     1     1      517            515         2
```

```
## 2     1     1      533            529         4
## 3     1     1      542            540         2
## 4     1     1      544            545        -1
## 5     1     1      554            600        -6
## # ... with 336,771 more rows
```

```
# Name starts with
select(flights, starts_with("d"))
```

```
## # A tibble: 336,776 x 5
##     day dep_time dep_delay  dest distance
##   <int>    <int>     <dbl> <chr>    <dbl>
## 1     1      517         2   IAH     1400
## 2     1      533         4   IAH     1416
## 3     1      542         2   MIA     1089
## 4     1      544        -1   BQN     1576
## 5     1      554        -6   ATL      762
## # ... with 336,771 more rows
```

```
# Name ends with
select(flights, ends_with("time"))
```

```
## # A tibble: 336,776 x 5
##   dep_time sched_dep_time arr_time sched_arr_time air_time
##      <int>          <int>    <int>          <int>    <dbl>
## 1      517            515      830            819      227
## 2      533            529      850            830      227
## 3      542            540      923            850      160
## 4      544            545     1004           1022      183
## 5      554            600      812            837      116
## # ... with 336,771 more rows
```

```
# Name contains
select(flights, contains("arr"))
```

```
## # A tibble: 336,776 x 4
##   arr_time sched_arr_time arr_delay carrier
##      <int>          <int>     <dbl>   <chr>
## 1      830            819        11      UA
## 2      850            830        20      UA
## 3      923            850        33      AA
## 4     1004           1022       -18      B6
## 5      812            837       -25      DL
## # ... with 336,771 more rows
```

```
# Name doesn't start with
select(flights, -starts_with("d"))
```

```
## # A tibble: 336,776 x 14
##    year month sched_dep_time arr_time sched_arr_time arr_delay carrier
##   <int> <int>          <int>    <int>          <int>     <dbl>   <chr>
## 1  2013     1            515      830            819        11      UA
## 2  2013     1            529      850            830        20      UA
## 3  2013     1            540      923            850        33      AA
## 4  2013     1            545     1004           1022       -18      B6
## 5  2013     1            600      812            837       -25      DL
## # ... with 336,771 more rows, and 7 more variables: flight <int>,
```

```
## #   tailnum <chr>, origin <chr>, air_time <dbl>, hour <dbl>, minute <dbl>,
## #   time_hour <dttm>
```

```
# Move column to the beginning
select(flights, flight, everything())
```

```
## # A tibble: 336,776 x 19
##   flight  year month   day dep_time sched_dep_time dep_delay arr_time
##    <int> <int> <int> <int>    <int>          <int>     <dbl>    <int>
## 1   1545  2013     1     1      517            515         2      830
## 2   1714  2013     1     1      533            529         4      850
## 3   1141  2013     1     1      542            540         2      923
## 4    725  2013     1     1      544            545        -1     1004
## 5    461  2013     1     1      554            600        -6      812
## # ... with 336,771 more rows, and 11 more variables: sched_arr_time <int>,
## #   arr_delay <dbl>, carrier <chr>, tailnum <chr>, origin <chr>,
## #   dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
## #   time_hour <dttm>
```

Look at the `?select` help page for a list of function to help you select multiple columns.

Verb composition with pipes

Traditioanlly, we combine functions via nesting, which works but is hard to read

```
select(filter(flights, dest=="DTW"), carrier)
```

```
## # A tibble: 9,384 x 1
##   carrier
##     <chr>
## 1      MQ
## 2      DL
## 3      DL
## 4      DL
## 5      DL
## # ... with 9,379 more rows
```

The `%>%` allows us to take an object, and pass it as the first parameter to another function. The above is the same as

```
flights %>%
  filter(dest=="DTW") %>%
  select(carrier)
```

```
## # A tibble: 9,384 x 1
##   carrier
##     <chr>
## 1      MQ
## 2      DL
## 3      DL
## 4      DL
## 5      DL
## # ... with 9,379 more rows
```

You can unroll any function with this operator

```
round(exp(sin(.5)),2)
```

```
## [1] 1.62
```

```
.5 %>% sin() %>% exp %>% round(2)
```

```
## [1] 1.62
```

Sorting Data

Use `arrange()` to sort data. You just specify the column names you want to sort by, use `desc()` to reverse the sort order for a given column.

```
flights %>% arrange(sched_dep_time)
```

```
## # A tibble: 336,776 x 19
##    year month   day dep_time sched_dep_time dep_delay arr_time
##   <int> <int> <int>    <int>          <int>     <dbl>    <int>
## 1  2013     7    27       NA            106        NA       NA
## 2  2013     1     2      458            500        -2      703
## 3  2013     1     3      458            500        -2      650
## 4  2013     1     4      456            500        -4      631
## 5  2013     1     5      458            500        -2      640
## # ... with 336,771 more rows, and 12 more variables: sched_arr_time <int>,
## #   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>
```

```
flights %>% arrange(month, desc(day))
```

```
## # A tibble: 336,776 x 19
##    year month   day dep_time sched_dep_time dep_delay arr_time
##   <int> <int> <int>    <int>          <int>     <dbl>    <int>
## 1  2013     1    31        1           2100       181      124
## 2  2013     1    31        4           2359         5      455
## 3  2013     1    31        7           2359         8      453
## 4  2013     1    31       12           2250        82      132
## 5  2013     1    31       26           2154       152      328
## # ... with 336,771 more rows, and 12 more variables: sched_arr_time <int>,
## #   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>
```

```
flights %>% arrange(desc(dep_time-sched_dep_time ))
```

```
## # A tibble: 336,776 x 19
##    year month   day dep_time sched_dep_time dep_delay arr_time
##   <int> <int> <int>    <int>          <int>     <dbl>    <int>
## 1  2013     3    17     2321            810       911      135
## 2  2013     7    22     2257            759       898      121
## 3  2013     2    10     2243            830       853      100
## 4  2013     2    19     2324           1016       788      114
## 5  2013     2    24     1921            615       786     2135
## # ... with 336,771 more rows, and 12 more variables: sched_arr_time <int>,
## #   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>
```

Creating New Variables

Use `mutate()` to create columns from existing columns or values

```
flights %>%
  mutate(speed = distance/(air_time/60)) %>%
  arrange(desc(speed)) %>%
  select(flight, speed)
```

```
## # A tibble: 336,776 x 2
##   flight    speed
##    <int>    <dbl>
## 1   1499 703.3846
## 2   4667 650.3226
## 3   4292 648.0000
## 4   3805 641.1429
## 5   1902 591.4286
## # ... with 336,771 more rows
```

You can create multiple columns by separating them with a comma; you can use any previously created columns as well

```
flights %>%
  mutate(
    dist_km = distance * 1.61,
    hours = air_time / 60,
    kph = dist_km/hours ) %>%
  select(flight, kph)
```

```
## # A tibble: 336,776 x 2
##   flight      kph
##    <int>    <dbl>
## 1   1545 595.7709
## 2   1714 602.5797
## 3   1141 657.4838
## 4    725 831.9213
## 5    461 634.5621
## # ... with 336,771 more rows
```

Use `summarize()` to collapse observations (only keeps columns for which you specified a summarization strategy)

```
flights %>%
  filter(!is.na(arr_delay)) %>%
  summarize(avg_arr_delay = mean(arr_delay))
```

```
## # A tibble: 1 x 1
##   avg_arr_delay
##           <dbl>
## 1      6.895377
```

Grouping Data

Perhaps the most powerful feature of `dplyr` is its grouping abilities. You can specify a column (or columns) for which `mutate()` and `summarize()` happen independently for each unique value in that column (or unique combination or values).

Using `summarize()` will reduce the total number of rows

```
flights %>%
  filter(!is.na(arr_delay)) %>%
  group_by(carrier) %>%
  summarize(avg_arr_delay = mean(arr_delay))
```

```
## # A tibble: 16 x 2
##    carrier avg_arr_delay
##      <chr>         <dbl>
## 1       9E     7.3796692
## 2       AA     0.3642909
## 3       AS    -9.9308886
## 4       B6     9.4579733
## 5       DL     1.6443409
## # ... with 11 more rows
```

Using `mutate()` will keep the same number of rows and won't drop any columns

```
flights %>%
  filter(!is.na(arr_delay)) %>%
  group_by(carrier) %>%
  mutate(avg_arr_delay = mean(arr_delay)) %>%
  select(carrier, arr_delay, avg_arr_delay)
```

```
## # A tibble: 327,346 x 3
## # Groups:   carrier [16]
##    carrier arr_delay avg_arr_delay
##      <chr>     <dbl>         <dbl>
## 1       UA        11     3.5580111
## 2       UA        20     3.5580111
## 3       AA        33     0.3642909
## 4       B6       -18     9.4579733
## 5       DL       -25     1.6443409
## # ... with 327,341 more rows
```

Joining data

When finding carriers with the largest flight delay, we were left with a carrier code rather than a carrier name; but who exactly is `9E`?

```
flights %>%
  filter(!is.na(arr_delay)) %>%
  group_by(carrier) %>%
  summarize(avg_arr_delay = mean(arr_delay))
```

```
## # A tibble: 16 x 2
##    carrier avg_arr_delay
##      <chr>         <dbl>
## 1       9E     7.3796692
```

```
## 2      AA     0.3642909
## 3      AS    -9.9308886
## 4      B6     9.4579733
## 5      DL     1.6443409
## # ... with 11 more rows
```

There is another table that has a lookup from carrier code to carrier name called airlines

```
airlines
```

```
## # A tibble: 16 x 2
##   carrier                   name
##     <chr>                  <chr>
## 1      9E      Endeavor Air Inc.
## 2      AA American Airlines Inc.
## 3      AS   Alaska Airlines Inc.
## 4      B6        JetBlue Airways
## 5      DL   Delta Air Lines Inc.
## # ... with 11 more rows
```

We can use `left_join` to merge in the carrier name

```
flights %>%
  filter(!is.na(arr_delay)) %>%
  group_by(carrier) %>%
  summarize(avg_arr_delay = mean(arr_delay)) %>%
  left_join(airlines)
```

```
## Joining, by = "carrier"
```

```
## # A tibble: 16 x 3
##   carrier avg_arr_delay                   name
##     <chr>         <dbl>                  <chr>
## 1      9E     7.3796692      Endeavor Air Inc.
## 2      AA     0.3642909 American Airlines Inc.
## 3      AS    -9.9308886   Alaska Airlines Inc.
## 4      B6     9.4579733        JetBlue Airways
## 5      DL     1.6443409   Delta Air Lines Inc.
## # ... with 11 more rows
```

Here we use two sample tables `x` and `y` to demonstrate the other types of joins

```
x <- tribble(
  ~key, ~xval,
  1, "x1",
  2, "x2",
  3, "x3")
y <- tribble(
  ~key, ~yval,
  1, "y1",
  2, "y2",
  4, "y3")
inner_join(x, y)
```

```
## Joining, by = "key"
```

```
## # A tibble: 2 x 3
##     key  xval  yval
##   <dbl> <chr> <chr>
```

```
## 1     1    x1    y1
## 2     2    x2    y2
```

```
left_join(x, y)
```

```
## Joining, by = "key"
```

```
## # A tibble: 3 x 3
##     key  xval  yval
##   <dbl> <chr> <chr>
## 1     1    x1    y1
## 2     2    x2    y2
## 3     3    x3  <NA>
```

```
right_join(x, y)
```

```
## Joining, by = "key"
```

```
## # A tibble: 3 x 3
##     key  xval  yval
##   <dbl> <chr> <chr>
## 1     1    x1    y1
## 2     2    x2    y2
## 3     4  <NA>    y3
```

```
full_join(x, y)
```

```
## Joining, by = "key"
```

```
## # A tibble: 4 x 3
##     key  xval  yval
##   <dbl> <chr> <chr>
## 1     1    x1    y1
## 2     2    x2    y2
## 3     3    x3  <NA>
## 4     4  <NA>    y3
```

And you can use non-merging joins to keep or drop rows that match keys from another table. Note that no new columns are added, just the rows of the input tables are filtered

```
z <- tribble(
  ~key, ~zval,
  1, "z1",
  3, "z2")
semi_join(x,z)
```

```
## Joining, by = "key"
```

```
## # A tibble: 2 x 2
##     key  xval
##   <dbl> <chr>
## 1     1    x1
## 2     3    x3
```

```
semi_join(y,z)
```

```
## Joining, by = "key"
```

```
## # A tibble: 1 x 2
##     key  yval
##   <dbl> <chr>
```

```
## 1     1    y1
```

```
anti_join(x,z)
```

```
## Joining, by = "key"
```

```
## # A tibble: 1 x 2
##     key  xval
##   <dbl> <chr>
## 1     2    x2
```

The join commands will join on all matching column names. You can more explicitly control this as well. The planes table has information about the aircraft used during the flight. It also happens to have a column named "year" indicating when the aircraft was built. When joining this data to flights, we only want to join on "tailnum" – not "tailnum" and "year".

```
flights %>%
  inner_join(planes) %>%
  nrow()  # wrong, only planes from 2013 are selected
```

```
## Joining, by = c("year", "tailnum")
```

```
## [1] 4630
```

```
flights %>%
  inner_join(planes, "tailnum") %>%
  nrow() # right
```

```
## [1] 284170
```

Subsetting functions

`distinct()` will return unique combinations of column values and nothing else

```
flights %>%
  distinct(tailnum, carrier)
```

```
## # A tibble: 4,067 x 2
##   carrier tailnum
##     <chr>   <chr>
## 1      UA  N14228
## 2      UA  N24211
## 3      AA  N619AA
## 4      B6  N804JB
## 5      DL  N668DN
## # ... with 4,062 more rows
```

The `count()` is like `distinct()` except it also returns the number of times each value was observed. It's basically a shortcut for `group_by() %>% summarize()`. For example

```
flights %>% count(carrier)
```

```
## # A tibble: 16 x 2
##   carrier     n
##     <chr> <int>
## 1      9E 18460
## 2      AA 32729
## 3      AS   714
## 4      B6 54635
```

```
## 5      DL 48110
## # ... with 11 more rows
```

```
flights %>% group_by(carrier) %>% summarize(n=n())
```

```
## # A tibble: 16 x 2
##   carrier     n
##     <chr> <int>
## 1      9E 18460
## 2      AA 32729
## 3      AS   714
## 4      B6 54635
## 5      DL 48110
## # ... with 11 more rows
```

`sample_n()` will randomly choose a set of rows from your table (different each time)

```
flights %>% sample_n(3)
```

```
## # A tibble: 3 x 19
##    year month   day dep_time sched_dep_time dep_delay arr_time
##   <int> <int> <int>    <int>          <int>     <dbl>    <int>
## 1  2013     9    24     1807           1815        -8     1938
## 2  2013     1    16      643            645        -2      811
## 3  2013    11    17     1709           1645        24     2013
## # ... with 12 more variables: sched_arr_time <int>, 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>
```

```
flights %>% sample_n(3)
```

```
## # A tibble: 3 x 19
##    year month   day dep_time sched_dep_time dep_delay arr_time
##   <int> <int> <int>    <int>          <int>     <dbl>    <int>
## 1  2013     4     9     1737           1735         2     2018
## 2  2013     9    27     1130           1130         0     1318
## 3  2013     7    12     1915           1840        35     2208
## # ... with 12 more variables: sched_arr_time <int>, 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>
```

You might also consider `anti_join` and `semi_join` to be subsetting commands rather than joining commands.

_at/_if/_all

The `summarize()`, `mutate()` and `group_by()` functions all have `_all()`, `_at()` and `_if()` variants that make it easier to apply the same function or functions to multiple columns.

`mutate_at()`, `summarize_at()` and `group_by_at()` allow you to choose columns in the same way you can do with select using the `vars()` helper function. This will take the mean of all columns that end in "_time"

```
flights %>%
  summarize_at(vars(ends_with("time")), mean, na.rm=T)
```

```
## # A tibble: 1 x 5
##   dep_time sched_dep_time arr_time sched_arr_time air_time
```

```
##      <dbl>          <dbl>    <dbl>          <dbl>    <dbl>
## 1  1349.11       1344.255 1502.055        1536.38 150.6865
```

`mutate_if()`, `summarize_if()` and `group_by_if()` allow you run a function on each column to choose only columns that meet a certain criteria. This can use useful for extracting columns of a certain class. Note you can also apply more than one function to these columns if you use the `funs()` helper function. This example will calculate the mean and variance for all numeric columns.

```
flights %>%
  summarize_if(is.numeric, funs(mean, var), na.rm=T)
```

```
## # A tibble: 1 x 28
##   year_mean month_mean day_mean dep_time_mean sched_dep_time_mean
##       <dbl>      <dbl>    <dbl>         <dbl>               <dbl>
## 1      2013    6.54851 15.71079       1349.11            1344.255
## # ... with 23 more variables: dep_delay_mean <dbl>, arr_time_mean <dbl>,
## #   sched_arr_time_mean <dbl>, arr_delay_mean <dbl>, flight_mean <dbl>,
## #   air_time_mean <dbl>, distance_mean <dbl>, hour_mean <dbl>,
## #   minute_mean <dbl>, year_var <dbl>, month_var <dbl>, day_var <dbl>,
## #   dep_time_var <dbl>, sched_dep_time_var <dbl>, dep_delay_var <dbl>,
## #   arr_time_var <dbl>, sched_arr_time_var <dbl>, arr_delay_var <dbl>,
## #   flight_var <dbl>, air_time_var <dbl>, distance_var <dbl>,
## #   hour_var <dbl>, minute_var <dbl>
```

The `_all()` versions of these functions will apply the same transformations to call non-grouped columns in the data source.

Other useful functions

The `lead()` and `lag()` functions are useful for selecting the next or previous values in a sequence (especially for time series data).

```
x<-1:5
lead(x)
```

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

```
lag(x)
```

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

The `coalesce()` function will return the first non-missing value from the vectors you pass to it. This is useful when you have multiple columns where only one column contains a value and you want to collapse them to a single vector

```
coalesce(c(NA,2,NA), c(1, NA, NA), 3)
```

```
## [1] 1 2 3
```

When using other `dplyr` verbs, the `n()` and `n_distinct()` functions will return the total number of observations or the number of unique observations respectively. In this example we look at the tail number for each plane to see how many total flights it took and also look at the number of distinct flight numbers that plane was a part of

```
flights %>%
  group_by(tailnum) %>%
  summarize(flights=n(), routes=n_distinct(flight))
```

```
## # A tibble: 4,044 x 3
```

```
##   tailnum flights routes
##     <chr>   <int>  <int>
## 1  D942DN       4      4
## 2  N0EGMQ     371    103
## 3  N10156     153    113
## 4  N102UW      48     37
## 5  N103US      46     26
## # ... with 4,039 more rows
```

The `recode()` function allows you to swap out certain values in a vector with different values.

```
recode(letters[1:5], b="boo")
```

```
## [1] "a"   "boo" "c"   "d"   "e"
```

The `case_when()` function allows more complex transformations than `recode()`. It's a good alternative to a bunch of nested `ifelse()` calls that you might need to use in base R. Each parameter should be a formula with a left-hand side value that evaulates to TRUE or FALSE and a right-hand side to return when that boolean value is TRUE. Only the value for the first TRUE is returned.

Here's a classic example of the "fizz buzz" problem where you are supposed to return the numbers 1-50 but replace all those values divisible by 5 with "fizz" and the values divisible by 7 with "buzz" and those divisible by both 5 and 7 by "fizz buzz"

```
x <- 1:50
case_when(
  x %% 35 == 0 ~ "fizz buzz",
  x %% 5 == 0 ~ "fizz",
  x %% 7 == 0 ~ "buzz",
  TRUE ~ as.character(x)
)
```

```
##  [1] "1"         "2"         "3"         "4"         "fizz"
##  [6] "6"         "buzz"      "8"         "9"         "fizz"
## [11] "11"        "12"        "13"        "buzz"      "fizz"
## [16] "16"        "17"        "18"        "19"        "fizz"
## [21] "buzz"      "22"        "23"        "24"        "fizz"
## [26] "26"        "27"        "buzz"      "29"        "fizz"
## [31] "31"        "32"        "33"        "34"        "fizz buzz"
## [36] "36"        "37"        "38"        "39"        "fizz"
## [41] "41"        "buzz"      "43"        "44"        "fizz"
## [46] "46"        "47"        "48"        "buzz"      "fizz"
```

Combining data frames

The `bind_rows()` and `bind_columns()` functions are alternatives to the base functions `rbind()` and `cbind()` that are list-friendly. Many times you end up with data.frames in a list that you want to combine in a single data.frame. These functions can help.

In this example, we have a list of two tibbles. We can combine them with `bind_rows`

```
x <- list(
  data_frame(a=1:2, z=letters[1:2]),
  data_frame(a=14:20, z=letters[14:20])
)
bind_rows(x)
```

```
## # A tibble: 9 x 2
```

```
##       a     z
##   <int> <chr>
## 1     1     a
## 2     2     b
## 3    14     n
## 4    15     o
## 5    16     p
## # ... with 4 more rows
```

```
bind_rows(x[[1]], x[[2]])
```

```
## # A tibble: 9 x 2
##       a     z
##   <int> <chr>
## 1     1     a
## 2     2     b
## 3    14     n
## 4    15     o
## 5    16     p
## # ... with 4 more rows
```

Programming with dplyr

Since dplyr uses non-standard evaluation to allow you to soecify data.frame column names without quotes, it can be tricky to write functions that use dply commands. Note that the first attempt at writting a function doesn't work

```
# Normal command, works fun
flights %>%
  group_by(carrier) %>%
  summarize(delay=mean(arr_delay, na.rm=T))
```

```
## # A tibble: 16 x 2
##    carrier      delay
##      <chr>      <dbl>
## 1       9E  7.3796692
## 2       AA  0.3642909
## 3       AS -9.9308886
## 4       B6  9.4579733
## 5       DL  1.6443409
## # ... with 11 more rows
```

```
# DOESN'T WORK
f <- function(x) {
  flights %>%
    group_by(x) %>%
    summarize(delay=mean(arr_delay, na.rm=T))
}
f(carrier)
```

```
## Error in grouped_df_impl(data, unname(vars), drop): Column `x` is unknown
```

The latest version of dplyr (0.7) introduced new way to write functions. Previously you would use the standard-evaluation version of functions that ended in an underscore (use `mutate_` rather than `mutate`); but the new version now uses "quosures" to allow you to pass column names. Here are two examples of functions that will work

```
f <- function(x) {
  flights %>% group_by(!!x) %>%
  summarize(delay = mean(arr_delay, na.rm=T))
}
f(quo(carrier))
```

```
## # A tibble: 16 x 2
##   carrier      delay
##     <chr>      <dbl>
## 1      9E  7.3796692
## 2      AA  0.3642909
## 3      AS -9.9308886
## 4      B6  9.4579733
## 5      DL  1.6443409
## # ... with 11 more rows
```

```
g <- function(x) {
  x <- enquo(x)
  flights %>% group_by(!!x) %>%
  summarize(delay = mean(arr_delay, na.rm=T))
}
g(carrier)
```

```
## # A tibble: 16 x 2
##   carrier      delay
##     <chr>      <dbl>
## 1      9E  7.3796692
## 2      AA  0.3642909
## 3      AS -9.9308886
## 4      B6  9.4579733
## 5      DL  1.6443409
## # ... with 11 more rows
```

We can either use `quo()` to create our own quosure with the column name, or we can use `enquo()` to turn a function parameter into a quosure.

Finally, in base R it's complicated to dynamically set the name of a parameter to a function (the name being the part to the left of the `=` in a call like `f(a=b)`). The latest dplyr functions now also allow you to use the value of a variable as a parameter name if you use `:=` rather than `=`. For example

```
h <- function(x) {
  x <- enquo(x)
  outname <- paste(quo_name(x), "delay", sep="_")
  flights %>% group_by(!!x) %>%
  summarize(!!outname := mean(arr_delay, na.rm=T))
}
h(carrier)
```

```
## # A tibble: 16 x 2
##   carrier carrier_delay
##     <chr>         <dbl>
## 1      9E     7.3796692
## 2      AA     0.3642909
## 3      AS    -9.9308886
## 4      B6     9.4579733
## 5      DL     1.6443409
```

```
## # ... with 11 more rows
```