Introduction to dplyr

Chao-Lung Yang
Department of Industrial Management
National Taiwan University of Science and Technology

Courtesy

- This powerpoint file summarize the “Introduction to dplyr” from https://cran.rstudio.com/web/packages/dplyr/vignettes/introduction.html where shows step by step learning on dplyr package which is very useful for manipulating data
- By compiling the mentioned webpage to powerpoint, the material is easier to introduce in the lecture

When working with data you must

- Figure out what you want to do
- Describe those tasks in the form of a computer program
- Execute the program

The dplyr package makes these steps fast and easy

- By constraining your options, it simplifies how you can think about common data manipulation tasks
- It provides simple “verbs”, functions that correspond to the most common data manipulation tasks, to help you translate those thoughts into code.
- It uses efficient data storage backends, so you spend less time waiting for the computer.

dplyr's basic set of tools

- Databases
  - Besides in-memory data frames, dplyr also connects to out-of-memory, remote databases. And by translating your R code into the appropriate SQL, it allows you to work with both types of data using the same set of tools.
- benchmark-baseball
  - see how dplyr compares to other tools for data manipulation on a realistic use case.
- window-functions
  - a window function is a variation on an aggregation function. Where an aggregate function uses n inputs to produce 1 output, a window function uses n inputs to produce n outputs.

Data: nycflights13

- To explore the basic data manipulation verbs of dplyr, we'll start with the built in nycflights13 data frame. This dataset contains all 336776 flights that departed from New York City in 2013. The data comes from the US Bureau of Transportation Statistics, and is documented in ?nycflights13

Prepare your nycflights13 data

```
install.packages("nycflights13")
library(nycflights13)
dim(flights)
head(flights)
```

```
> library(nycflights13)
> dim(flights)
[1] 336776     16
>
> head(flights)
  year month day dep_time dep_delay arr_time arr_delay carrier tailnum
1 2013     1   1      517         2      830        11      UA  N14228
2 2013     1   1      533         4      850        20      UA  N24211
3 2013     1   1      542         2      923        33      AA  N619AA
4 2013     1   1      544        -1     1004       -18      B6  N804JB
5 2013     1   1      554        -6      812       -25      DL  N668DN
6 2013     1   1      554        -4      740        12      UA  N39463
  flight origin dest air_time distance hour minute
1   1545    EWR  IAH      227     1400    5     17
2   1714    LGA  IAH      227     1416    5     33
3   1141    JFK  MIA      160     1089    5     42
4    725    JFK  BQN      183     1576    5     44
5    461    LGA  ATL      116      762    5     54
6   1696    EWR  ORD      150      719    5     54
>
```

Large data: tbl_df

- dplyr can work with data frames as is, but if you're dealing with large data, it's worthwhile to convert them to a tbl_df: this is a wrapper around a data frame that won't accidentally print a lot of data to the screen.

SINGLE TABLE VERBS

Single table verbs

- Dplyr aims to provide a function for each basic verb of data manipulation:
  - filter() (and slice())
  - arrange()
  - select() (and rename())
  - distinct()
  - mutate() (and transmute())
  - summarise()
  - sample_n() and sample_frac()
- If you've used plyr before, many of these will be familiar.

Filter rows with filter()

- filter() allows you to select a subset of rows in a data frame. The first argument is the name of the data frame. The second and subsequent arguments are the expressions that filter the data frame:
- For example, we can select all flights on January 1st with:

Filter rows with filter()

filter(flights, month == 1, day == 1)

```
> filter(flights, month == 1, day == 1)
Source: local data frame [842 x 16]

    year month   day dep_time dep_delay arr_time arr_delay carrier tailnum flight
   (int) (int) (int)    (int)     (dbl)    (int)     (dbl)   (chr)   (chr)  (int)
1   2013     1     1      517         2      830        11      UA  N14228   1545
2   2013     1     1      533         4      850        20      UA  N24211   1714
3   2013     1     1      542         2      923        33      AA  N619AA   1141
4   2013     1     1      544        -1     1004       -18      B6  N804JB    725
5   2013     1     1      554        -6      812       -25      DL  N668DN    461
6   2013     1     1      554        -4      740        12      UA  N39463   1696
7   2013     1     1      555        -5      913        19      B6  N516JB    507
8   2013     1     1      557        -3      709       -14      EV  N829AS   5708
9   2013     1     1      557        -3      838        -8      B6  N593JB     79
10  2013     1     1      558        -2      753         8      AA  N3ALAA    301
..   ...   ...   ...      ...       ...      ...       ...     ...     ...    ...
Variables not shown: origin (chr), dest (chr), air_time (dbl), distance (dbl), hour
   (dbl), minute (dbl)
> |
```

This is equivalent to the more verbose code in base R:

flights[flights$month == 1 & flights$day == 1,]

filter() vs. subset()

- filter() works similarly to subset() except that you can give it any number of filtering conditions, which are joined together with & (not && which is easy to do accidentally!). You can also use other boolean operators:

```
filter(flights, month == 1 | month == 2)
```

slices()

- To select rows by position, use slice():

slice(flights, 1:10)

```
> slice(flights, 1:10)
Source: local data frame [10 x 16]

    year month   day dep_time dep_delay arr_time arr_delay carrier tailnum flight
   (int) (int) (int)    (int)     (dbl)    (int)     (dbl)   (chr)   (chr)  (int)
1   2013     1     1      517         2      830        11      UA  N14228   1545
2   2013     1     1      533         4      850        20      UA  N24211   1714
3   2013     1     1      542         2      923        33      AA  N619AA   1141
4   2013     1     1      544        -1     1004       -18      B6  N804JB    725
5   2013     1     1      554        -6      812       -25      DL  N668DN    461
6   2013     1     1      554        -4      740        12      UA  N39463   1696
7   2013     1     1      555        -5      913        19      B6  N516JB    507
8   2013     1     1      557        -3      709       -14      EV  N829AS   5708
9   2013     1     1      557        -3      838        -8      B6  N593JB     79
10  2013     1     1      558        -2      753         8      AA  N3ALAA    301
Variables not shown: origin (chr), dest (chr), air_time (dbl), distance (dbl), hour
  (dbl), minute (dbl)
> |
```

Arrange rows with arrange()

- arrange() works similarly to filter() except that instead of filtering or selecting rows, it reorders them
- It takes a data frame, and a set of column names (or more complicated expressions) to order by
- If you provide more than one column name, each additional column will be used to break ties in the values of preceding columns:

arrange()

arrange(flights, year, month, day)

```
> arrange(flights, year, month, day)
Source: local data frame [336,776 x 16]

    year month   day dep_time dep_delay arr_time arr_delay carrier tailnum flight
   (int) (int) (int)    (int)     (dbl)    (int)     (dbl)   (chr)   (chr)  (int)
1   2013     1     1      517         2      830        11      UA  N14228   1545
2   2013     1     1      533         4      850        20      UA  N24211   1714
3   2013     1     1      542         2      923        33      AA  N619AA   1141
4   2013     1     1      544        -1     1004       -18      B6  N804JB    725
5   2013     1     1      554        -6      812       -25      DL  N668DN    461
6   2013     1     1      554        -4      740        12      UA  N39463   1696
7   2013     1     1      555        -5      913        19      B6  N516JB    507
8   2013     1     1      557        -3      709       -14      EV  N829AS   5708
9   2013     1     1      557        -3      838        -8      B6  N593JB     79
10  2013     1     1      558        -2      753         8      AA  N3ALAA    301
..   ...   ...   ...      ...       ...      ...       ...     ...     ...    ...
Variables not shown: origin (chr), dest (chr), air_time (dbl), distance (dbl), hour
   (dbl), minute (dbl)
>
```

arrange() and desc()

- Use desc() to order a column in descending order:

arrange(flights, desc(arr_delay))

```
> arrange(flights, desc(arr_delay))
Source: local data frame [336,776 x 16]

    year month   day dep_time dep_delay arr_time arr_delay carrier tailnum flight
   (int) (int) (int)    (int)     (dbl)    (int)     (dbl)   (chr)   (chr)  (int)
1   2013     1     9      641      1301     1242      1272      HA  N384HA     51
2   2013     6    15     1432      1137     1607      1127      MQ  N504MQ   3535
3   2013     1    10     1121      1126     1239      1109      MQ  N517MQ   3695
4   2013     9    20     1139      1014     1457      1007      AA  N338AA    177
5   2013     7    22      845      1005     1044       989      MQ  N665MQ   3075
6   2013     4    10     1100       960     1342       931      DL  N959DL   2391
7   2013     3    17     2321       911      135       915      DL  N927DA   2119
8   2013     7    22     2257       898      121       895      DL  N6716C   2047
9   2013    12     5      756       896     1058       878      AA  N5DMAA    172
10  2013     5     3     1133       878     1250       875      MQ  N523MQ   3744
..   ...   ...   ...      ...       ...      ...       ...     ...     ...    ...
Variables not shown: origin (chr), dest (chr), air_time (dbl), distance (dbl), hour
   (dbl), minute (dbl)
> |
```

- The previous code is equivalent to:

```
flights[order(flights$year, flights$month, flights$day), ]
flights[order(desc(flights$arr_delay)), ]
```

Select columns with select()

- Often you work with large datasets with many columns but only a few are actually of interest to you
- select() allows you to rapidly zoom in on a useful subset using operations that usually only work on numeric variable positions:

select()

Select columns by name

select(flights, year, month, day)

```
> select(flights, year, month, day)
Source: local data frame [336,776 x 3]

    year month   day
   (int) (int) (int)
1   2013     1     1
2   2013     1     1
3   2013     1     1
4   2013     1     1
5   2013     1     1
6   2013     1     1
7   2013     1     1
8   2013     1     1
9   2013     1     1
10  2013     1     1
..   ...   ...   ...
> 
```

select()

- # Select all columns between year and day (inclusive)

select(flights, year:day)

```
> select(flights, year:day)
Source: local data frame [336,776 x 3]

    year month   day
   (int) (int) (int)
1   2013     1     1
2   2013     1     1
3   2013     1     1
4   2013     1     1
5   2013     1     1
6   2013     1     1
7   2013     1     1
8   2013     1     1
9   2013     1     1
10  2013     1     1
..   ...   ...   ...
> |
```

select()

- # Select all columns except those from year to day (inclusive)

select(flights, -(year:day))

```
> select(flights, -(year:day))
Source: local data frame [336,776 x 13]

   dep_time dep_delay arr_time arr_delay carrier tailnum flight origin  dest air_time
      (int)     (dbl)    (int)     (dbl)   (chr)   (chr)  (int)  (chr) (chr)    (dbl)
1       517         2      830        11      UA  N14228   1545    EWR   IAH      227
2       533         4      850        20      UA  N24211   1714    LGA   IAH      227
3       542         2      923        33      AA  N619AA   1141    JFK   MIA      160
4       544        -1     1004       -18      B6  N804JB    725    JFK   BQN      183
5       554        -6      812       -25      DL  N668DN    461    LGA   ATL      116
6       554        -4      740        12      UA  N39463   1696    EWR   ORD      150
7       555        -5      913        19      B6  N516JB    507    EWR   FLL      158
8       557        -3      709       -14      EV  N829AS   5708    LGA   IAD       53
9       557        -3      838        -8      B6  N593JB     79    JFK   MCO      140
10      558        -2      753         8      AA  N3ALAA    301    LGA   ORD      138
..      ...       ...      ...       ...     ...     ...    ...    ...   ...      ...
Variables not shown: distance (dbl), hour (dbl), minute (dbl)
>
```

- There are a number of helper functions you can use within select(), like starts_with(), ends_with(), matches() and contains()
- These let you quickly match larger blocks of variables that meet some criterion.
  See ?select for more details.

rename column in select()

- You can rename variables with rename() by using named arguments:

rename(flights, tail_num = tailnum)

```
> rename(flights, tail_num = tailnum)
Source: local data frame [336,776 x 16]

    year month   day dep_time dep_delay arr_time arr_delay carrier tail_num flight
   (int) (int) (int)    (int)     (dbl)    (int)     (dbl)   (chr)    (chr)  (int)
1   2013     1     1      517         2      830        11      UA   N14228   1545
2   2013     1     1      533         4      850        20      UA   N24211   1714
3   2013     1     1      542         2      923        33      AA   N619AA   1141
4   2013     1     1      544        -1     1004       -18      B6   N804JB    725
5   2013     1     1      554        -6      812       -25      DL   N668DN    461
6   2013     1     1      554        -4      740        12      UA   N39463   1696
7   2013     1     1      555        -5      913        19      B6   N516JB    507
8   2013     1     1      557        -3      709       -14      EV   N829AS   5708
9   2013     1     1      557        -3      838        -8      B6   N593JB     79
10  2013     1     1      558        -2      753         8      AA   N3ALAA    301
..   ...   ...   ...      ...       ...      ...       ...     ...      ...    ...
Variables not shown: origin (chr), dest (chr), air_time (dbl), distance (dbl), hour
   (dbl), minute (dbl)
>
```

Extract distinct (unique) rows

- A common use of select() is to find the values of a set of variables. This is particularly useful in conjunction with the distinct() verb which only returns the unique values in a table.

distinct(select(flights, tailnum))

```
> distinct(select(flights, tailnum))
Source: local data frame [4,044 x 1]

   tailnum
     (chr)
1   N14228
2   N24211
3   N619AA
4   N804JB
5   N668DN
6   N39463
7   N516JB
8   N829AS
9   N593JB
10  N3ALAA
..     ...
> |
```

distinct(select(flights, origin, dest))

```
> distinct(select(flights, origin, dest))
Source: local data frame [224 x 2]

   origin  dest
    (chr) (chr)
1     EWR   IAH
2     LGA   IAH
3     JFK   MIA
4     JFK   BQN
5     LGA   ATL
6     EWR   ORD
7     EWR   FLL
8     LGA   IAD
9     JFK   MCO
10    LGA   ORD
..    ...   ...
> 
```

This is very similar to base::unique() but should be much faster.

Add new columns with mutate()

- Besides selecting sets of existing columns, it's often useful to add new columns that are functions of existing columns. This is the job of mutate():

mutate()

mutate(flights, gain = arr_delay - dep_delay, speed = distance / air_time * 60)

```
> mutate(flights,
+   gain = arr_delay - dep_delay,
+   speed = distance / air_time * 60)
Source: local data frame [336,776 x 18]

    year month   day dep_time dep_delay arr_time arr_delay carrier tailnum flight
   (int) (int) (int)    (int)     (dbl)    (int)     (dbl)   (chr)   (chr)  (int)
1   2013     1     1      517         2      830        11      UA  N14228   1545
2   2013     1     1      533         4      850        20      UA  N24211   1714
3   2013     1     1      542         2      923        33      AA  N619AA   1141
4   2013     1     1      544        -1     1004       -18      B6  N804JB    725
5   2013     1     1      554        -6      812       -25      DL  N668DN    461
6   2013     1     1      554        -4      740        12      UA  N39463   1696
7   2013     1     1      555        -5      913        19      B6  N516JB    507
8   2013     1     1      557        -3      709       -14      EV  N829AS   5708
9   2013     1     1      557        -3      838        -8      B6  N593JB     79
10  2013     1     1      558        -2      753         8      AA  N3ALAA    301
..   ...   ...   ...      ...       ...      ...       ...     ...     ...    ...
Variables not shown: origin (chr), dest (chr), air_time (dbl), distance (dbl), hour
  (dbl), minute (dbl), gain (dbl), speed (dbl)
>
```

- mutate allows you to refer to columns that you've just created:

mutate(flights, gain = arr_delay - dep_delay, gain_per_hour = gain / (air_time / 60))

```
> mutate(flights,
+   gain = arr_delay - dep_delay,
+   gain_per_hour = gain / (air_time / 60)
+ )
Source: local data frame [336,776 x 18]

    year month   day dep_time dep_delay arr_time arr_delay carrier tailnum flight
   (int) (int) (int)    (int)     (dbl)    (int)     (dbl)   (chr)   (chr)  (int)
1   2013     1     1      517         2      830        11      UA  N14228   1545
2   2013     1     1      533         4      850        20      UA  N24211   1714
3   2013     1     1      542         2      923        33      AA  N619AA   1141
4   2013     1     1      544        -1     1004       -18      B6  N804JB    725
5   2013     1     1      554        -6      812       -25      DL  N668DN    461
6   2013     1     1      554        -4      740        12      UA  N39463   1696
7   2013     1     1      555        -5      913        19      B6  N516JB    507
8   2013     1     1      557        -3      709       -14      EV  N829AS   5708
9   2013     1     1      557        -3      838        -8      B6  N593JB     79
10  2013     1     1      558        -2      753         8      AA  N3ALAA    301
..   ...   ...   ...      ...       ...      ...       ...     ...     ...    ...
Variables not shown: origin (chr), dest (chr), air_time (dbl), distance (dbl), hour
  (dbl), minute (dbl), gain (dbl), gain_per_hour (dbl)
>
```

transmute()

- If you only want to keep the new variables, use transmute():

transmute(flights, gain = arr_delay - dep_delay, gain_per_hour = gain / (air_time / 60))

```
> transmute(flights,
+   gain = arr_delay - dep_delay,
+   gain_per_hour = gain / (air_time / 60)
+ )
Source: local data frame [336,776 x 2]

    gain gain_per_hour
   (dbl)         (dbl)
1     9      2.378855
2    16      4.229075
3    31     11.625000
4   -17     -5.573770
5   -19     -9.827586
6    16      6.400000
7    24      9.113924
8   -11    -12.452830
9    -5     -2.142857
10   10      4.347826
..  ...           ...
> |
```

Summarise values with summarise()

- The last verb is summarise(). It collapses a data frame to a single row

summarise(flights, delay = mean(dep_delay, na.rm = TRUE))

```
> summarise(flights,
+   delay = mean(dep_delay, na.rm = TRUE))
Source: local data frame [1 x 1]

     delay
     (dbl)
1 12.63907
> 
```

Randomly sample rows with sample_n() and sample_frac()

- You can use sample_n() and sample_frac() to take a random sample of rows
- use sample_n() for a fixed number and sample_frac() for a fixed fraction.

sample_n(flights, 10)

```
>
> sample_n(flights, 10)
Source: local data frame [10 x 16]

    year month   day dep_time dep_delay arr_time arr_delay carrier tailnum flight
   (int) (int) (int)    (int)     (dbl)    (int)     (dbl)   (chr)   (chr)  (int)
1   2013     7    27      657        -3      952       -11      DL  N679DA   1415
2   2013     9    18     1324        -1     1617       -13      UA  N411UA    295
3   2013    11    15      715       -15     1026       -29      VX  N835VA    183
4   2013     6    16     1425        -5     1545       -13      FL  N993AT    721
5   2013     1    10     1052        -8     1354       -30      DL  N976DL   2044
6   2013     5     9     1818        13     1952        -2      UA  N77530   1053
7   2013     4    27     1433        -7     1612       -18      MQ  N738MQ   4429
8   2013     3     2     1834        -6     2138       -22      DL  N37700   1643
9   2013     8     2     1656        66     1916        71      WN  N768SW    262
10  2013     3    15     1239        -6     1345        -5      AA  N3GAAA   1850
Variables not shown: origin (chr), dest (chr), air_time (dbl), distance (dbl), hour
  (dbl), minute (dbl)
```

sample_frac(flights, 0.01)

```
> sample_frac(flights, 0.01)
Source: local data frame [3,368 x 16]

    year month   day dep_time dep_delay arr_time arr_delay carrier tailnum flight
   (int) (int) (int)    (int)     (dbl)    (int)     (dbl)   (chr)   (chr)  (int)
1   2013     8    28      750       -12      905       -25      EV  N717EV   5209
2   2013     9    19     1956        -4     2144       -25      EV  N12552   4293
3   2013    10     8      830         5     1020        -2      B6  N249JB    219
4   2013    12    29     2400       420      302       397      AA  N3GMAA   2379
5   2013     6    29      822        -1     1051       -49      UA  N75432   1151
6   2013    11    29     1608        -7     1855       -30      AA  N3KRAA    211
7   2013     7    15     1449         4     1617         1      UA  N39416   1290
8   2013     6    10     2148        78       10       113      B6  N267JB    917
9   2013     3    23     1502        17     1815        27      UA  N38727   1186
10  2013     4    13     1227        -3     1451        11      EV  N750EV   5148
..   ...   ...   ...      ...       ...      ...       ...     ...     ...    ...
Variables not shown: origin (chr), dest (chr), air_time (dbl), distance (dbl), hour
   (dbl), minute (dbl)
>
```

Use replace = TRUE to perform a bootstrap sample. If needed, you can weight the sample with the weight argument.

Commonalities

- You may have noticed that the syntax and function of all these verbs are very similar:
  - The first argument is a data frame.
  - The subsequent arguments describe what to do with the data frame. Notice that you can refer to columns in the data frame directly without using $.
  - The result is a new data frame
- Together these properties make it easy to chain together multiple simple steps to achieve a complex result.

- At the most basic level, you can only alter a tidy data frame in five useful ways: you can
  - reorder the rows (arrange())
  - pick observations and variables of interest (filter() and select())
  - add new variables that are functions of existing variables (mutate())
  - collapse many values to a summary (summarise())
- The remainder of the language comes from applying the five functions to different types of data. For example, how these functions work with grouped data.

GROUPED OPERATIONS

- These verbs are useful on their own, but they become really powerful when you apply them to groups of observations within a dataset.
- In dplyr, you do this by with the group_by() function
- It breaks down a dataset into specified groups of rows
- When you then apply the verbs above on the resulting object they'll be automatically applied "by group".
- Most importantly, all this is achieved by using the same exact syntax you'd use with an ungrouped object.

- In the following example, we split the complete dataset into individual planes and then summarise each plane by counting the number of flights (count = n()) and computing the average distance (dist = mean(Distance, na.rm = TRUE)) and arrival delay (delay = mean(ArrDelay, na.rm = TRUE)). We then use ggplot2 to display the output.

```
by_tailnum <- group_by(flights, tailnum)
delay <- summarise(by_tailnum,
 count = n(),
 dist = mean(distance, na.rm = TRUE),
 delay = mean(arr_delay, na.rm = TRUE))
delay <- filter(delay, count > 20, dist < 2000)
```

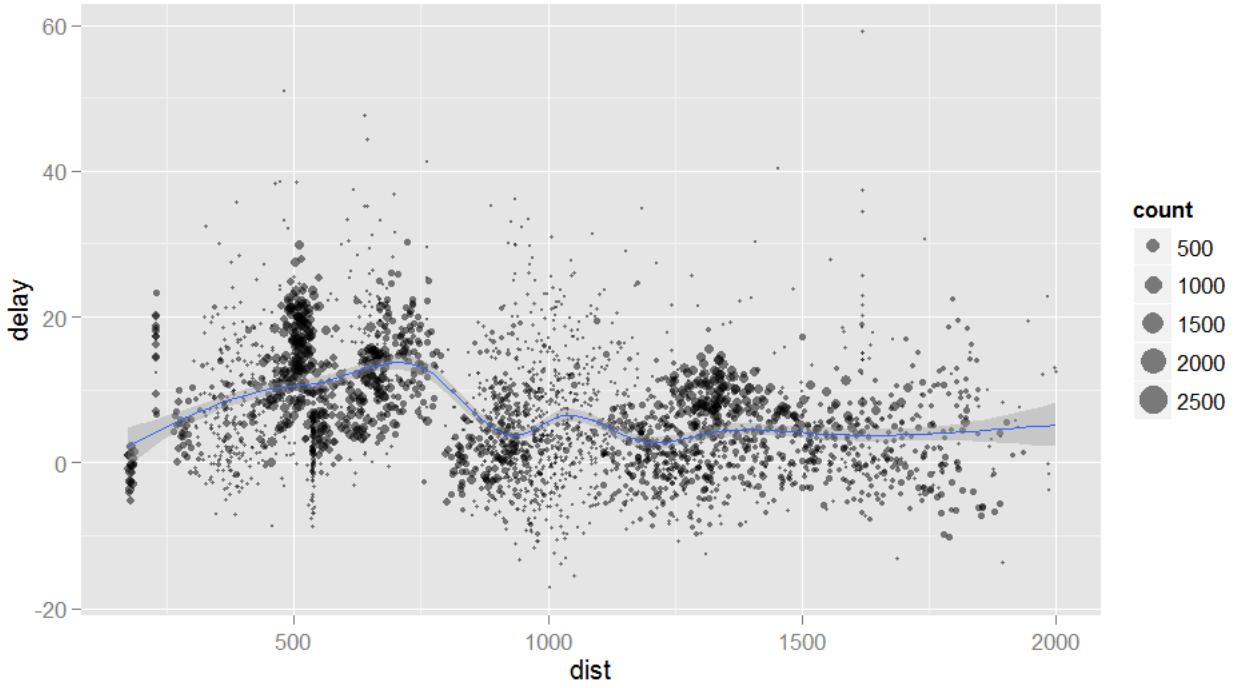
```
> delay <- summarise(by_tailnum,
+   count = n(),
+   dist = mean(distance, na.rm = TRUE),
+   delay = mean(arr_delay, na.rm = TRUE))
> delay <- filter(delay, count > 20, dist < 2000)
>
> delay
Source: local data frame [2,962 x 4]

   tailnum count     dist      delay
     (chr) (int)    (dbl)      (dbl)
1           2512 710.2576        NaN
2   N0EGMQ   371 676.1887  9.9829545
3   N10156   153 757.9477 12.7172414
4   N102UW    48 535.8750  2.9375000
5   N103US    46 535.1957 -6.9347826
6   N104UW    47 535.2553  1.8043478
7   N10575   289 519.7024 20.6914498
8   N105UW    45 524.8444 -0.2666667
9   N107US    41 528.7073 -5.7317073
10  N108UW    60 534.5000 -1.2500000
..     ...   ...      ...        ...
> 
```

Interestingly, the average delay is only slightly related to the

average distance flown by a plane.

```
ggplot(delay, aes(dist, delay)) + geom_point(aes(size = count), alpha = 1/2) + geom_smooth() + scale_size_area()
```



- You use summarise() with **aggregate functions**, which take a vector of values and return a single number. There are many useful examples of such functions in base R like min(), max(), mean(), sum(), sd(), median(), and IQR(). dplyr provides a handful of others:
  - n(): the number of observations in the current group
  - n_distinct(x):the number of unique values in x.
  - first(x), last(x) and nth(x, n) - these work similarly to x[1], x[length(x)], and x[n] but give you more control over the result if the value is missing.

- For example, we could use these to find the number of planes and the number of flights that go to each possible destination:

```
destinations <- group_by(flights, dest)
summarise(destinations,
 planes = n_distinct(tailnum),
 flights = n()
)
```

```
> destinations <- group_by(flights, dest)
> summarise(destinations,
+   planes = n_distinct(tailnum),
+   flights = n()
+ )
Source: local data frame [105 x 3]

    dest planes flights
   (chr)  (int)   (int)
1    ABQ    108     254
2    ACK     58     265
3    ALB    172     439
4    ANC      6       8
5    ATL   1180   17215
6    AUS    993    2439
7    AVL    159     275
8    BDL    186     443
9    BGR     46     375
10   BHM     45     297
..   ...    ...     ...
> 
```

- You can save the result in a new variable

```
> t <- summarise(destinations,
+   planes = n_distinct(tailnum),
+   flights = n()
+ )
>
> t
Source: local data frame [105 x 3]

    dest planes flights
   (chr)  (int)   (int)
1    ABQ    108     254
2    ACK     58     265
3    ALB    172     439
4    ANC      6       8
5    ATL   1180   17215
6    AUS    993    2439
7    AVL    159     275
8    BDL    186     443
9    BGR     46     375
10   BHM     45     297
..   ...    ...     ...
```

- Check type of t object

```
> class(t)
[1] "tbl_df"     "tbl"        "data.frame"
```

- Check all of data by change t object to data.frame

```
> data.frame(t)
    dest planes flights
1    ABQ    108     254
2    ACK     58     265
3    ALB    172     439
4    ANC      6       8
5    ATL   1180   17215
6    AUS    993    2439
7    AVL    159     275
8    BDL    186     443
9    BGR     46     375
10   BHM     45     297
11   BNA    963    6333
12   BOS   1308   15508
13   BQN    268     896
14   BTV    390    2589
15   BUF    504    4681
16   BUR    120     371
17   BWI    632    1781
18   BZN     32      36
```

- When you group by multiple variables, each summary peels off one level of the grouping. That makes it easy to progressively roll-up a dataset:

```
> daily <- group_by(flights, year, month, day)
> (per_day   <- summarise(daily, flights = n()))
Source: local data frame [365 x 4]
Groups: year, month [?]

    year month   day flights
   (int) (int) (int)   (int)
1   2013     1     1     842
2   2013     1     2     943
3   2013     1     3     914
4   2013     1     4     915
5   2013     1     5     720
6   2013     1     6     832
7   2013     1     7     933
8   2013     1     8     899
9   2013     1     9     902
10  2013     1    10     932
..   ...   ...   ...     ...
> |
```

```
> (per_month <- summarise(per_day, flights = sum(flights)))
Source: local data frame [12 x 3]
Groups: year [?]

    year month flights
   (int) (int)   (int)
1   2013     1   27004
2   2013     2   24951
3   2013     3   28834
4   2013     4   28330
5   2013     5   28796
6   2013     6   28243
7   2013     7   29425
8   2013     8   29327
9   2013     9   27574
10  2013    10   28889
11  2013    11   27268
12  2013    12   28135
> 
```

```
> (per_year  <- summarise(per_month, flights = sum(flights)))
Source: local data frame [1 x 2]

   year flights
  (int)   (int)
1  2013  336776
> 
```

Chaining

- The dplyr API is functional in the sense that function calls don't have side-effects
- You must always save their results
- This doesn't lead to particularly elegant code, especially if you want to do many operations at once

- You either have to do it step-by-step:

```
a1 <- group_by(flights, year, month, day)
a2 <- select(a1, arr_delay, dep_delay)
a3 <- summarise(a2,
  arr = mean(arr_delay, na.rm = TRUE),
  dep = mean(dep_delay, na.rm = TRUE))
a4 <- filter(a3, arr > 30 | dep > 30)
```

```
> a4
Source: local data frame [49 x 5]
Groups: year, month [11]

    year month   day      arr      dep
   (int) (int) (int)    (dbl)    (dbl)
1   2013     1    16 34.24736 24.61287
2   2013     1    31 32.60285 28.65836
3   2013     2    11 36.29009 39.07360
4   2013     2    27 31.25249 37.76327
5   2013     3     8 85.86216 83.53692
6   2013     3    18 41.29189 30.11796
7   2013     4    10 38.41231 33.02368
8   2013     4    12 36.04814 34.83843
9   2013     4    18 36.02848 34.91536
10  2013     4    19 47.91170 46.12783
..   ...   ...   ...      ...      ...
```

- Or if you don't want to save the intermediate results, you need to wrap the function calls inside each other:

```
filter(
  summarise(
    select(
      group_by(flights, year, month, day),
      arr_delay, dep_delay
    ),
    arr = mean(arr_delay, na.rm = TRUE),
    dep = mean(dep_delay, na.rm = TRUE)
  ),
  arr > 30 | dep > 30
)
```

```
> filter(
+   summarise(
+     select(
+       group_by(flights, year, month, day),
+       arr_delay, dep_delay
+     ),
+     arr = mean(arr_delay, na.rm = TRUE),
+     dep = mean(dep_delay, na.rm = TRUE)
+   ),
+   arr > 30 | dep > 30
+ )
Source: local data frame [49 x 5]
Groups: year, month [11]

    year month   day      arr      dep
   (int) (int) (int)    (dbl)    (dbl)
1   2013     1    16 34.24736 24.61287
2   2013     1    31 32.60285 28.65836
3   2013     2    11 36.29009 39.07360
4   2013     2    27 31.25249 37.76327
5   2013     3     8 85.86216 83.53692
6   2013     3    18 41.29189 30.11796
7   2013     4    10 38.41231 33.02368
8   2013     4    12 36.04814 34.83843
9   2013     4    18 36.02848 34.91536
10  2013     4    19 47.91170 46.12783
..   ...   ...   ...      ...      ...
> |
```

- This is difficult to read because the order of the operations is from inside to out
- Thus, the arguments are a long way away from the function
- To get around this problem, dplyr provides the %>% operator
- x %>% f(y) turns into f(x, y) so you can use it to rewrite multiple operations that you can read left-to-right, top-to-bottom:

New Expression by %>%

```
flights %>%
  group_by(year, month, day) %>%
  select(arr_delay, dep_delay) %>%
  summarise(
    arr = mean(arr_delay, na.rm = TRUE),
    dep = mean(dep_delay, na.rm = TRUE)
  ) %>%
  filter(arr > 30 | dep > 30)
```

```
> flights %>%
+   group_by(year, month, day) %>%
+   select(arr_delay, dep_delay) %>%
+   summarise(
+     arr = mean(arr_delay, na.rm = TRUE),
+     dep = mean(dep_delay, na.rm = TRUE)
+   ) %>%
+   filter(arr > 30 | dep > 30)
Source: local data frame [49 x 5]
Groups: year, month [11]

    year month   day      arr      dep
   (int) (int) (int)    (dbl)    (dbl)
1   2013     1    16 34.24736 24.61287
2   2013     1    31 32.60285 28.65836
3   2013     2    11 36.29009 39.07360
4   2013     2    27 31.25249 37.76327
5   2013     3     8 85.86216 83.53692
6   2013     3    18 41.29189 30.11796
7   2013     4    10 38.41231 33.02368
8   2013     4    12 36.04814 34.83843
9   2013     4    18 36.02848 34.91536
10  2013     4    19 47.91170 46.12783
..   ...   ...   ...      ...      ...
> |
```