

Data transformation

Yifan Jin

09/02/2020

Introduction

Visualisation is an important tool for insight generation, but it is rare that you get the data in exactly the right form you need. We may need to create, summarise, rename, reorder variables.

Prerequisites

```
# install.packages("nycflights13")
library(nycflights13)
library(tidyverse)

## -- Attaching packages ----- tidyverse 1.3.0 --

## v ggplot2 3.2.1      v purrr  0.3.3
## v tibble  2.1.3      v dplyr  0.8.3
## v tidyr   1.0.0      v stringr 1.4.0
## v readr   1.3.1      v forcats 0.4.0

## -- Conflicts ----- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()     masks stats::lag()
```

nycflights13

```
flights

## # A tibble: 336,776 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 336,766 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 <dtm>
int: integers
dbl: doubles, or real numbers
chr: character vectors, or strings
dtm: date-times (a date+ a time)
lgl: logical, vectors that contain only TRUE or FALSE
fctr: factors, which R uses to represent categorical variables with fixed possible values
date: dates
```

dplyr basics

`filter()`: pick observations by their values

`arrange()`: reorder the rows

`select()`: pick variables by their names

`mutate()`: create new variables with functions of existing variable

`summarise()`: collapse many values down to single summary

All verbs work similarly:

1. The first argument is a data frame.
2. The subsequent arguments describe what to do with the data frame, using the variable names (without quotes)
3. The result is a new data frame

Filter rows with `filter()`

`filter()` allows you to subset observations based on their values. The first argument is the name of the data frame. The second and subsequent arguments are the expressions that filter the data frame.

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

```
## # A tibble: 842 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 832 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 <dtm>
```

```
jan1<-filter(flights,month==1,day==1)

(dec25<-filter(flights,month==12,day==25))
```

```
## # A tibble: 719 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    12    25     456           500          -4       649           651
## 2  2013    12    25     524           515           9       805           814
## 3  2013    12    25     542           540           2       832           850
## 4  2013    12    25     546           550          -4      1022          1027
## 5  2013    12    25     556           600          -4       730           745
## 6  2013    12    25     557           600          -3       743           752
## 7  2013    12    25     557           600          -3       818           831
## 8  2013    12    25     559           600          -1       855           856
## 9  2013    12    25     559           600          -1       849           855
## 10 2013    12    25      600           600           0       850           846
## # ... with 709 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 <dtm>
# brackets that we can print the result
```

Comparisons

Comparison operator: >, >=, <,<=, != and ==

```
# filter(flights,month=1) This is false since we need to use ==
sqrt(2)^2==2
```

```
## [1] FALSE
```

```
1/49*49==1
```

```
## [1] FALSE
```

This result is surprising, when we compare two numbers, one is floating, one is integer

#Instead of relying on `==`, we use near()

```
near(sqrt(2)^2,2)
```

```
## [1] TRUE
```

```
near(1/49*49,1)
```

```
## [1] TRUE
```

Logical operators

&: and

|: or

!: not

```
filter(flights, month==11|month==12)
```

```
## # A tibble: 55,403 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     1       5             2359           6         352           345
## 2  2013    11     1      35             2250          105        123          2356
## 3  2013    11     1     455             500           -5        641           651
## 4  2013    11     1     539             545           -6        856           827
## 5  2013    11     1     542             545           -3        831           855
## 6  2013    11     1     549             600          -11        912           923
## 7  2013    11     1     550             600          -10        705           659
## 8  2013    11     1     554             600           -6        659           701
## 9  2013    11     1     554             600           -6        826           827
## 10 2013    11     1     554             600           -6        749           751
## # ... with 55,393 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 <dtm>
```

```
# or
```

```
nov_dec<-filter(flights, month %in% c(11,12))
```

Missing values

```
NA>5
```

```
## [1] NA
```

```
10==NA
```

```
## [1] NA
```

```
NA+10
```

```
## [1] NA
```

```
NA/2
```

```
## [1] NA
```

```
NA==NA
```

```
## [1] NA
```

```
X<-NA
```

```
Y<-NA
```

```
X==Y
```

```
## [1] NA
```

```
df<-tibble(x=c(1,NA,3))
```

```
filter(df, x>1)
```

```
## # A tibble: 1 x 1
```

```
##       x
```

```
##   <dbl>
```

```
## 1     3
```

```
filter(df,is.na(x)|x>1)
```

```
## # A tibble: 2 x 1
##       x
##   <dbl>
## 1    NA
## 2     3
```

Exercises

1. Find all flights that had an arrival delay of two or more hours

Answer

```
filter(flights,arr_delay>120)
```

```
## # A tibble: 10,034 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     811             630          101    1047             830
## 2  2013     1     1     848             1835         853    1001             1950
## 3  2013     1     1     957             733          144    1056             853
## 4  2013     1     1    1114             900          134    1447             1222
## 5  2013     1     1    1505             1310          115    1638             1431
## 6  2013     1     1    1525             1340          105    1831             1626
## 7  2013     1     1    1549             1445           64    1912             1656
## 8  2013     1     1    1558             1359          119    1718             1515
## 9  2013     1     1    1732             1630           62    2028             1825
## 10 2013     1     1    1803             1620          103    2008             1750
## # ... with 10,024 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 <dtm>
```

2. Find all flights that Flew to Houston (IAH or HOU)

Answer

```
filter(flights,dest=="IAH"|dest=="HOU")
```

```
## # A tibble: 9,313 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     623             627          -4     933             932
## 4  2013     1     1     728             732          -4    1041            1038
## 5  2013     1     1     739             739           0    1104            1038
## 6  2013     1     1     908             908           0    1228            1219
## 7  2013     1     1    1028            1026           2    1350            1339
## 8  2013     1     1    1044            1045          -1    1352            1351
## 9  2013     1     1    1114             900          134    1447            1222
## 10 2013     1     1    1205            1200           5    1503            1505
## # ... with 9,303 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 <dtm>
```

3. Find all flights that were operated by United, American, or Delta

Answer

```
filter(flights, carrier=="UA" | carrier=="AA" | carrier=="DL")
```

```
## # A tibble: 139,504 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     554           600          -6     812           837
## 5  2013     1     1     554           558          -4     740           728
## 6  2013     1     1     558           600          -2     753           745
## 7  2013     1     1     558           600          -2     924           917
## 8  2013     1     1     558           600          -2     923           937
## 9  2013     1     1     559           600          -1     941           910
##10  2013     1     1     559           600          -1     854           902
## # ... with 139,494 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 <dtm>
```

4. Departed in summer (July, August, and September)

Answer

```
filter(flights, month==7 | month==8 | month==9)
```

```
## # A tibble: 86,326 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     7     1       1           2029          212     236           2359
## 2  2013     7     1       2           2359           3     344           344
## 3  2013     7     1      29           2245          104     151             1
## 4  2013     7     1      43           2130          193     322             14
## 5  2013     7     1      44           2150          174     300            100
## 6  2013     7     1      46           2051          235     304           2358
## 7  2013     7     1      48           2001          287     308           2305
## 8  2013     7     1      58           2155          183     335             43
## 9  2013     7     1     100           2146          194     327             30
##10  2013     7     1     100           2245          135     337            135
## # ... with 86,316 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 <dtm>
```

5. Arrived more than two hours late, but didn't leave late

```
filter(flights, arr_delay>120 | dep_delay<=0)
```

```
## # A tibble: 210,094 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     544           545          -1    1004           1022
## 2  2013     1     1     554           600          -6     812           837
## 3  2013     1     1     554           558          -4     740           728
## 4  2013     1     1     555           600          -5     913           854
## 5  2013     1     1     557           600          -3     709           723
```

```
## 6 2013 1 1 557 600 -3 838 846
## 7 2013 1 1 558 600 -2 753 745
## 8 2013 1 1 558 600 -2 849 851
## 9 2013 1 1 558 600 -2 853 856
## 10 2013 1 1 558 600 -2 924 917
## # ... with 210,084 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 <dtm>
```

6. Were delayed by at least an hours late, but made up over 30 minutes in flight

Answer

```
filter(flights, arr_delay < dep_delay - 30 | dep_delay >= 60)
```

```
## # A tibble: 43,165 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     701           700         1     1123          1154
## 2 2013     1     1     811           630        101     1047           830
## 3 2013     1     1     820           820         0     1249          1329
## 4 2013     1     1     826           715         71     1136          1045
## 5 2013     1     1     840           845        -5     1311          1350
## 6 2013     1     1     848          1835        853     1001          1950
## 7 2013     1     1     857           851         6     1157          1222
## 8 2013     1     1     909           810         59     1331          1315
## 9 2013     1     1     957           733        144     1056           853
## 10 2013     1     1    1025           951         34     1258          1302
## # ... with 43,155 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 <dtm>
```

7. Departed between midnight and 6am (inclusive)

Answer

```
filter(flights, dep_time >= 000 & dep_time <= 600)
```

```
## # A tibble: 9,344 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 9,334 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 <dtm>
```

8. Another useful dplyr filtering helper is between(). What does it do?

Answer

```
filter(flights,between(dep_time,000,600))
```

```
## # A tibble: 9,344 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 9,334 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 <dtm>
```

```
filter(flights,is.na(dep_time))
```

```
## # A tibble: 8,255 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      NA           1630        NA      NA           1815
## 2  2013     1     1      NA           1935        NA      NA           2240
## 3  2013     1     1      NA           1500        NA      NA           1825
## 4  2013     1     1      NA           600         NA      NA           901
## 5  2013     1     2      NA           1540        NA      NA           1747
## 6  2013     1     2      NA           1620        NA      NA           1746
## 7  2013     1     2      NA           1355        NA      NA           1459
## 8  2013     1     2      NA           1420        NA      NA           1644
## 9  2013     1     2      NA           1321        NA      NA           1536
## 10 2013     1     2      NA           1545        NA      NA           1910
## # ... with 8,245 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 <dtm>
```

Arrange rows with arrange()

arrange() works similarly to filter() except that instead of selecting rows, it changes their order.

```
arrange(flights,year,month,day)
```

```
## # A tibble: 336,776 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 336,766 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 <dtm>
```

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

```
arrange(flights, desc(dep_delay))
```

```
## # A tibble: 336,776 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     9     641             900         1301    1242         1530
## 2 2013     6    15    1432            1935         1137    1607         2120
## 3 2013     1    10    1121            1635         1126    1239         1810
## 4 2013     9    20    1139            1845         1014    1457         2210
## 5 2013     7    22     845            1600         1005    1044         1815
## 6 2013     4    10    1100            1900          960    1342         2211
## 7 2013     3    17    2321             810          911     135         1020
## 8 2013     6    27     959            1900          899    1236         2226
## 9 2013     7    22    2257             759          898     121         1026
## 10 2013    12     5     756            1700          896    1058         2020
## # ... with 336,766 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 <dtm>
```

Missing values are always sorted at the end:

```
df<-tibble(x=c(5,2,NA))
arrange(df,x)
```

```
## # A tibble: 3 x 1
##       x
##   <dbl>
## 1     2
## 2     5
## 3    NA
```

```
arrange(df, desc(x))
```

```
## # A tibble: 3 x 1
##       x
##   <dbl>
## 1     5
## 2     2
## 3    NA
```

Exercises

1. How could you use `arrange()` to sort all missing values to the start? (hint: use `is.na()`)

Answer

```
arrange(df, desc(is.na(x)))
```

```
## # A tibble: 3 x 1
```

```
##      x
##   <dbl>
## 1    NA
## 2     5
## 3     2
```

2. Sort flights to find the most delayed flights. Find the flights that left earliest

Answer

```
arrange(flights, desc(dep_delay))
```

```
## # A tibble: 336,776 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     9     641             900      1301      1242          1530
## 2  2013     6    15    1432            1935      1137      1607          2120
## 3  2013     1    10    1121            1635      1126      1239          1810
## 4  2013     9    20    1139            1845      1014      1457          2210
## 5  2013     7    22     845            1600      1005      1044          1815
## 6  2013     4    10    1100            1900       960      1342          2211
## 7  2013     3    17    2321             810       911       135          1020
## 8  2013     6    27     959            1900       899      1236          2226
## 9  2013     7    22    2257             759       898       121          1026
## 10 2013    12     5     756            1700       896      1058          2020
## # ... with 336,766 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 <dtm>
```

```
arrange(flights, dep_delay)
```

```
## # A tibble: 336,776 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    12     7    2040            2123       -43        40          2352
## 2  2013     2     3    2022            2055       -33       2240          2338
## 3  2013    11    10    1408            1440       -32       1549          1559
## 4  2013     1    11    1900            1930       -30       2233          2243
## 5  2013     1    29    1703            1730       -27       1947          1957
## 6  2013     8     9     729             755       -26       1002           955
## 7  2013    10    23    1907            1932       -25       2143          2143
## 8  2013     3    30    2030            2055       -25       2213          2250
## 9  2013     3     2    1431            1455       -24       1601          1631
## 10 2013     5     5     934             958       -24       1225          1309
## # ... with 336,766 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 <dtm>
```

3. Sort flights to find the fastest (highest speed) flights.

Answer

```
arrange(flights, desc(distance/air_time))
```

```
## # A tibble: 336,776 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    25    1709            1700         9      1923          1937
```

```
## 2 2013 7 2 1558 1513 45 1745 1719
## 3 2013 5 13 2040 2025 15 2225 2226
## 4 2013 3 23 1914 1910 4 2045 2043
## 5 2013 1 12 1559 1600 -1 1849 1917
## 6 2013 11 17 650 655 -5 1059 1150
## 7 2013 2 21 2355 2358 -3 412 438
## 8 2013 11 17 759 800 -1 1212 1255
## 9 2013 11 16 2003 1925 38 17 36
## 10 2013 11 16 2349 2359 -10 402 440
## # ... with 336,766 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 <dtm>
```

4. Which flights travelled the farthest? Which travelled the shortest?

Answer

```
arrange(flights, desc(distance))
```

```
## # A tibble: 336,776 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 1 857 900 -3 1516 1530
## 2 2013 1 2 909 900 9 1525 1530
## 3 2013 1 3 914 900 14 1504 1530
## 4 2013 1 4 900 900 0 1516 1530
## 5 2013 1 5 858 900 -2 1519 1530
## 6 2013 1 6 1019 900 79 1558 1530
## 7 2013 1 7 1042 900 102 1620 1530
## 8 2013 1 8 901 900 1 1504 1530
## 9 2013 1 9 641 900 1301 1242 1530
## 10 2013 1 10 859 900 -1 1449 1530
## # ... with 336,766 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 <dtm>
```

```
arrange(flights, distance)
```

```
## # A tibble: 336,776 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 7 27 NA 106 NA NA 245
## 2 2013 1 3 2127 2129 -2 2222 2224
## 3 2013 1 4 1240 1200 40 1333 1306
## 4 2013 1 4 1829 1615 134 1937 1721
## 5 2013 1 4 2128 2129 -1 2218 2224
## 6 2013 1 5 1155 1200 -5 1241 1306
## 7 2013 1 6 2125 2129 -4 2224 2224
## 8 2013 1 7 2124 2129 -5 2212 2224
## 9 2013 1 8 2127 2130 -3 2304 2225
## 10 2013 1 9 2126 2129 -3 2217 2224
## # ... with 336,766 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 <dtm>
```

Select columns with select()

It is not uncommon to get datasets with hundreds or even thousands of variables. In this case, the first challenge is to narrowing in on the variables you are actually interested in. We use `select()`.

```
select(flights,year,month,day) #directly type columns that want to stay
```

```
## # A tibble: 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
## # ... with 336,766 more rows
```

```
select(flights,year:day) # use : to select variables between left and right
```

```
## # A tibble: 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
## # ... with 336,766 more rows
```

```
select(flights,-(year:day)) # delete the columns that dont want
```

```
## # A tibble: 336,776 x 16
##   dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay carrier
##   <int>         <int>      <dbl>   <int>         <int>        <dbl> <chr>
## 1     517           515         2     830           819         11 UA
## 2     533           529         4     850           830         20 UA
## 3     542           540         2     923           850         33 AA
## 4     544           545        -1    1004          1022        -18 B6
## 5     554           600        -6     812           837        -25 DL
## 6     554           558        -4     740           728         12 UA
## 7     555           600        -5     913           854         19 B6
## 8     557           600        -3     709           723        -14 EV
## 9     557           600        -3     838           846         -8 B6
## 10    558           600        -2     753           745          8 AA
## # ... with 336,766 more rows, and 9 more variables: flight <int>,
## #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
```

```
## # hour <dbl>, minute <dbl>, time_hour <dtm>
```

There are a number of helper functions you can see within `select()`:

1. `starts_with("abc")`: matches names that begin with “abc”
2. `ends_with("xyz")`: matches names that end with “xyz”
3. `contains("ijk")`: matches names that contain “ijk”
4. `matches("(.)\\1")`: selects variables that match a regular expression. This one matches any variables that contain repeated characters. You will learn more about regular expressions in strings.
5. `num_range("x",1:3)`: matches `x1`, `x2` and `x3`.

`select()` can be used to rename variables, but it is rarely useful because it drops all variables not explicitly mentioned. Instead, `rename()`, which is a variant of `select()` that keeps all the variables that are not explicitly mentioned:

```
rename(flights,tail_num=tailnum)
```

```
## # A tibble: 336,776 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 336,766 more rows, and 11 more variables: arr_delay <dbl>,
## #   carrier <chr>, flight <int>, tail_num <chr>, origin <chr>, dest <chr>,
## #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dtm>
```

Another option is to use `select()` in conjunction with the `everything()` helper.

```
select(flights,time_hour,air_time,everything())
```

```
## # A tibble: 336,776 x 19
##   time_hour          air_time year month   day dep_time sched_dep_time
##   <dtm>             <dbl> <int> <int> <int>   <int>         <int>
## 1 2013-01-01 05:00:00      227  2013     1     1     517             515
## 2 2013-01-01 05:00:00      227  2013     1     1     533             529
## 3 2013-01-01 05:00:00      160  2013     1     1     542             540
## 4 2013-01-01 05:00:00      183  2013     1     1     544             545
## 5 2013-01-01 06:00:00      116  2013     1     1     554             600
## 6 2013-01-01 05:00:00      150  2013     1     1     554             558
## 7 2013-01-01 06:00:00      158  2013     1     1     555             600
## 8 2013-01-01 06:00:00       53  2013     1     1     557             600
## 9 2013-01-01 06:00:00      140  2013     1     1     557             600
## 10 2013-01-01 06:00:00      138  2013     1     1     558             600
## # ... with 336,766 more rows, and 12 more variables: dep_delay <dbl>,
## #   arr_time <int>, sched_arr_time <int>, arr_delay <dbl>, carrier <chr>,
## #   flight <int>, tailnum <chr>, origin <chr>, dest <chr>, distance <dbl>,
## #   hour <dbl>, minute <dbl>
```

Exercises

1. Brainstorm as many ways as possible to select `dep_time`, `dep_delay`, `arr_time`, and `arr_delay` from `flights`

Answer

```
# 5!=5*4*3*2*1
```

2. What happens if you include the name of a variable multiple times in a `select()` call?

Answer

```
select(flights,distance,distance)
```

```
## # A tibble: 336,776 x 1
##   distance
##   <dbl>
## 1     1400
## 2     1416
## 3     1089
## 4     1576
## 5       762
## 6       719
## 7     1065
## 8       229
## 9       944
## 10      733
## # ... with 336,766 more rows
```

Only one column will be showed.

3. What does the `one_of()` function do? Why might it be helpful in conjunction with this vector?

Answer

```
vars<-c("year","month","day","dep_delay","arr_delay")
select(flights,one_of(vars))
```

```
## # A tibble: 336,776 x 5
##   year month   day dep_delay arr_delay
##   <int> <int> <int>   <dbl>   <dbl>
## 1  2013     1     1         2        11
## 2  2013     1     1         4        20
## 3  2013     1     1         2        33
## 4  2013     1     1        -1       -18
## 5  2013     1     1        -6       -25
## 6  2013     1     1        -4        12
## 7  2013     1     1        -5        19
## 8  2013     1     1        -3       -14
## 9  2013     1     1        -3        -8
## 10 2013     1     1        -2         8
## # ... with 336,766 more rows
```

```
# one_of(): Matches variable names in a character vector.
```

4. Does the result of running the following code surprise you? How do the select helpers deal with case by default? How can you change that default?

Answer

```
select(flights, contains("TIME"))
```

```
## # A tibble: 336,776 x 6
##   dep_time sched_dep_time arr_time sched_arr_time air_time time_hour
##   <int>      <int>      <int>      <int>      <dbl> <dtm>
## 1     517         515        830         819      227 2013-01-01 05:00:00
## 2     533         529        850         830      227 2013-01-01 05:00:00
## 3     542         540        923         850      160 2013-01-01 05:00:00
## 4     544         545       1004       1022      183 2013-01-01 05:00:00
## 5     554         600        812         837      116 2013-01-01 06:00:00
## 6     554         558        740         728      150 2013-01-01 05:00:00
## 7     555         600        913         854      158 2013-01-01 06:00:00
## 8     557         600        709         723       53 2013-01-01 06:00:00
## 9     557         600        838         846      140 2013-01-01 06:00:00
## 10    558         600        753         745      138 2013-01-01 06:00:00
## # ... with 336,766 more rows
```

Including all variables which name contains time.

Add new variables with mutate()

mutate(): building new variable with existing variables

```
flights_sml <- select(flights, year:day,
                      ends_with("delay"),
                      distance,
                      air_time)

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

```
## # A tibble: 336,776 x 9
##   year month   day dep_delay arr_delay distance air_time gain speed
##   <int> <int> <int>    <dbl>    <dbl>    <dbl>    <dbl> <dbl> <dbl>
## 1  2013     1     1         2        11    1400     227     -9   370.
## 2  2013     1     1         4        20    1416     227    -16   374.
## 3  2013     1     1         2        33    1089     160    -31   408.
## 4  2013     1     1        -1       -18    1576     183     17   517.
## 5  2013     1     1        -6       -25     762     116     19   394.
## 6  2013     1     1        -4        12     719     150    -16   288.
## 7  2013     1     1        -5        19    1065     158    -24   404.
## 8  2013     1     1        -3       -14     229      53     11   259.
## 9  2013     1     1        -3        -8     944     140      5   405.
## 10 2013     1     1        -2         8     733     138    -10   319.
## # ... with 336,766 more rows
```

You can refer to columns that you have just created

```
mutate(flights_sml,
       gain = dep_delay - arr_delay,
       hours = air_time / 60,
       gain_per_hours = gain / hours)
```

```
## # A tibble: 336,776 x 10
```

```
##      year month   day dep_delay arr_delay distance air_time  gain hours
##      <int> <int> <int>    <dbl>    <dbl>    <dbl>    <dbl> <dbl> <dbl>
##  1  2013     1     1         2        11     1400     227    -9  3.78
##  2  2013     1     1         4        20     1416     227   -16  3.78
##  3  2013     1     1         2        33     1089     160   -31  2.67
##  4  2013     1     1        -1       -18     1576     183    17  3.05
##  5  2013     1     1        -6       -25      762     116    19  1.93
##  6  2013     1     1        -4        12      719     150   -16  2.5
##  7  2013     1     1        -5        19     1065     158   -24  2.63
##  8  2013     1     1        -3       -14      229      53    11  0.883
##  9  2013     1     1        -3        -8      944     140     5  2.33
## 10  2013     1     1        -2         8      733     138   -10  2.3
## # ... with 336,766 more rows, and 1 more variable: gain_per_hours <dbl>
```

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

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

```
## # A tibble: 336,776 x 3
##      gain hours gain_per_hour
##      <dbl> <dbl>    <dbl>
##  1    -9  3.78    -2.38
##  2   -16  3.78    -4.23
##  3   -31  2.67   -11.6
##  4    17  3.05     5.57
##  5    19  1.93     9.83
##  6   -16  2.5    -6.4
##  7   -24  2.63   -9.11
##  8    11  0.883    12.5
##  9     5  2.33     2.14
## 10   -10  2.3    -4.35
## # ... with 336,766 more rows
```

Useful creation functions

There are many functions for creating new variables that you can use with `mutate()`. The key property is that the function must be vectorised: it must take a vector of values as input, return a vector with the same number of values as output. There is no way to list every possible function that you might use, but there is a selection of functions that are frequently useful:

1. Arithmetic operators: `+`, `-`, `*`, `/`, `^`. These are all vectorised using the so called “recycling rules”. `x/sum(x)`- calculate the proportion of a total, `y-mean(y)`- calculate the difference from the mean
2. Modular arithmetic: `%/%` (integer division) and `%%` (remainder), where `x==y*(x%/%y)+(x%%y)`. Modular arithmetic is a handy tool because it allows you to break integers up into pieces.

```
transmute(flights,
  dep_time,
  hour=dep_time%/% 100,
  minute=dep_time %%100)
```

```
## # A tibble: 336,776 x 3
##      dep_time  hour minute
```



```
##           <int> <dbl> <dbl>
##  1         517      5      17
##  2         533      5      33
##  3         542      5      42
##  4         544      5      44
##  5         554      5      54
##  6         554      5      54
##  7         555      5      55
##  8         557      5      57
##  9         557      5      57
## 10         558      5      58
## # ... with 336,766 more rows
```

3. Cumulative and rolling aggregates: R provides functions for running sums, products, mins and maxes: `cumsum()`, `cumprod()`, `cummin()`, `cummax()`; and `dplyr` provides `cummean()` for cumulative means.

```
x<-c(1,2,3,4,5,6,7,8,9,10)
cumsum(x)
```

```
## [1]  1  3  6 10 15 21 28 36 45 55
```

```
cummean(x)
```

```
## [1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5
```

4. Logical comparisons, `<`, `<=`, `>`, `>=`, `!=`, and `==`, which you learned earlier.

5. Ranking: there are a number of ranking functions, but you should start with `min_rank()`, use `desc` to get largest to smallest.

```
y<-c(1,2,2,NA,3,4)
min_rank(y)
```

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

```
min_rank(desc(y))
```

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

Grouped summarises with `summarise()`