Data Manipulation with dplyr

Vahan Aslanyan

Data Wrangling and Manipulation

- Makes data accessible to users
- Creates final dataset(s) for projects (analysis,report,etc)
- Create new variables or summaries
- Rename the variables or observations

Data Frames

- Data frames are the most common structures used for analyses
- Identifiable columns (with preferably non-empty headers) containing variables of interest
- Rows containing observed values of variables of interest
- Mixed formats allowed (strings, dates, numeric values); matrices can be transformed into dataframe

Example

```r
# to install: install.packages("nycflights13")
# to install: install.packages("tidyverse")
library(nycflights13)
library("dplyr")
top_n(flights,10) #from nycflight13 package
```

```
# A tibble: 12 × 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    31       13           2359        14      439            437
 2  2013    12    31       18           2359        19      449            444
 3  2013    12    31       26           2245       101      129           2353
 4  2013    12    31     2218           2219        -1      315            304
 5  2013    12    31     2235           2245       -10     2351           2355
 6  2013    12    31     2245           2250        -5     2359           2356
 7  2013    12    31     2310           2255        15        7           2356
 8  2013    12    31     2321           2250        31       46              8
 9  2013    12    31     2328           2330        -2      412            409
10  2013    12    31     2332           2245        47       58              3
11  2013    12    31     2355           2359        -4      430            440
12  2013    12    31     2356           2359        -3      436            445
```

Data manipulation–Reasons

- Original data is not ready to use
- Maybe you are interested in a subset of variables
- Maybe you are only interested in specific observations
- Maybe you need to create new variables (morning arrivals vs evening arrivals)
- Maybe you need to fix the encoding of the variables (arr_time is an integer now, we need to have it as time)

Data manipulation–Motivation

- Certain effort goes into data manipulation
- 80/20 rule: 80% of time is spent cleaning up data, 20%– doing analyses or generating insights
- We want a clean readable code
- We want our steps to be intuitive

Dplyr package

- “Grammar of data manipulation”
- Provides a consistent set of verbs that help solve the most common data manipulation challenges
- https://dplyr.tidyverse.org/
- Cheat sheet: https://github.com/rstudio/cheatsheets/blob/master/data-transformation.pdf

Dplyr basics

- Pick observations by their values (filter())
- Reorder the rows (arrange())
- Pick variables by their names (select())
- Create new variables with functions of existing variables (mutate())
- Collapse many values down to a single summary (summarise())

These can all be used in conjunction with group_by() which changes the scope of each function from operating on the entire dataset to operating on it group-by-group.

Filter rows with filter()

filter() allows to subset observations based on their values. For example we can select all flights on January first with:

```
flights%>%
  filter(month==1,day==1)->flight_Jan01
top_n(flight_Jan01,10)
```

```
# A tibble: 14 × 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     2158           2200        -2     2254           2307
 2  2013     1     1     2217           2229       -12      249            315
 3  2013     1     1     2224           2200        24     2324           2316
 4  2013     1     1     2240           2245        -5     2340           2356
 5  2013     1     1     2250           2255        -5     2352           2359
 6  2013     1     1     2302           2200        62     2342           2253
 7  2013     1     1     2306           2245        21       28              5
 8  2013     1     1     2307           2245        22       32           2357
 9  2013     1     1     2310           2255        15       24             15
10  2013     1     1     2323           2200        83       22           2313
11  2013     1     1     2327           2250        37       32           2359
12  2013     1     1     2353           2359        -6      425            445
```

Comparisons

R useses the standard suite for comparisons

- > greater than
- >= greater than or equal to
- < less than
- <= less than or equal to
- == equal
- != not equal

Exercise: Early bird

A traveler always has meetings at 10 am so she has to arrive before 8:30 to be there on time. She is planning a trip in May and wants to arrive in STL before then. Create a dataframe flights_830 that contains all May flights that arrive at STL before 8:30

Exercise: Early bird

```
 1  flights%>%
 2    filter(month==5, arr_time<=830,dest=="STL")->flights_830
 3  flights_830
```

```
# A tibble: 36 × 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     1      559            600        -1      725            745
 2  2013     5     2      604            600         4      728            745
 3  2013     5     3      600            600         0      733            745
 4  2013     5     4      624            630        -6      735            815
 5  2013     5     5      559            600        -1      708            745
 6  2013     5     6      557            600        -3      714            745
 7  2013     5     7      603            600         3      720            745
 8  2013     5     8      601            600         1      721            745
 9  2013     5     8     2314           2124       110       43           2305
10  2013     5     9      605            600         5      735            745
# … with 26 more rows, and 11 more variables: arr_delay <dbl>, carrier <chr>,
#   flight <int>, tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>,
```

Logical operators

- & and
- | or
- ! not

Now lets get all flights in November and December:

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

```
# A tibble: 55,403 × 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>,
```

Logical operators

```
 1  flights%>%filter(month==11| month ==12)
```

```
# A tibble: 55,403 × 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>,
```

Logical operators

```
 1  filter(flights, month %in% c(11, 12))
```

```
# A tibble: 55,403 × 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>,
```

Logical operators

If we wanted to find flights that weren't delayed (on arrival or departure) by more than two hours, you could use either of the following two filters:

```
1 filter(flights, !(arr_delay > 120 | dep_delay > 120))
```

```
# A tibble: 316,050 × 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 316,040 more rows, and 11 more variables: arr_delay <dbl>,
#   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
```

Logical operators

```
 1  filter(flights, arr_delay <= 120, dep_delay <= 120)
```

```
# A tibble: 316,050 × 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 316,040 more rows, and 11 more variables: arr_delay <dbl>,
#   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
```

Missing values

Missing values NAs make comparison tricky. If we want to determine if a value is missing, use is.na():

```
1 df <- tibble(x = c(1, NA, 3))
2 df%>%filter(is.na(x) | x > 1)
```

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

Arrange rows with arrange()

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

```
1 flights%>%arrange(year, month, day)
```

```
# A tibble: 336,776 × 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>,
```

Arrange rows with arrange()- Continued

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

```
1 flights%>%arrange(desc(dep_delay))
```

```
# A tibble: 336,776 × 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>,
```

Arrange rows with arrange()- Continued

- Missing values are always sorted at the end:

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

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

Select columns with select()

- Narrow in on the variables we are interested in
- Select() allows to rapidly zoom in on a useful subset using operations based on the names of the variables

```
1 # Select columns by name
2 select(flights, year,month, day)
```

```
# A tibble: 336,776 × 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
```

Select columns with select()

```
 1 # Select all columns between year and day (inclusive)
 2 select(flights, year:day)
```

```
# A tibble: 336,776 × 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 columns with select()

```
1 # Select all columns except those from year to day (inclusive)
2 select(flights, -(year:day))
```

```
# A tibble: 336,776 × 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>,
```

Select columns with select()

- There are a number of helper functions we can use within select():
- starts_with(“abc”): matches names that begin with “abc”
- ends_with(“xyz”): matches names that end with “xyz”
- contains(“ijk”): matches names that contain “ijk”
- num_range(“x”, 1:3): matches x1, x2, and x3

Select columns with select()

- everything() is useful for rearranging columns

```
# A tibble: 336,776 × 19
   time_hour           air_time  year month   day dep_time sched_dep_time
   <dttm>                 <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>,
```

Rename variables with rename()

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

```
# A tibble: 336,776 × 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>,
```

Add new variable with mutate()

- Sometimes we need to create new columns that are function of existing columns
- mutate() always add new columns at the end of the dataset

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

Add new variable with mutate()

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

```
# A tibble: 336,776 × 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
```

Add new variable with mutate()

We can refer to columns that we have just created

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

```
# A tibble: 336,776 × 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_hour <dbl>
```

Add new variable with mutate()

We can use transmute() if we only want to keep the new variables

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

```
# A tibble: 336,776 × 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
```

Grouped summaries with summarise()

- summarise() collapses a data frame to a single row

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

```
# A tibble: 1 × 1
  delay
  <dbl>
1  12.6
```

Grouped summaries with summarise()

- summarise() has limited usefulness unless we pair it with group_by()
- group_by() changes the unit of analysis from the complete dataset to individual groups

```
1 by_day <- group_by(flights, year, month, day)
2 summarise(by_day, delay = mean(dep_delay, na.rm = TRUE))
```

```
# A tibble: 365 × 4
# Groups:   year, month [12]
    year month   day delay
   <int> <int> <int> <dbl>
 1  2013     1     1 11.5
 2  2013     1     2 13.9
 3  2013     1     3 11.0
 4  2013     1     4  8.95
```

```
 5  2013     1     5  5.73
 6  2013     1     6  7.15
 7  2013     1     7  5.42
 8  2013     1     8  2.55
 9  2013     1     9  2.28
10  2013     1    10  2.84
# … with 355 more rows
```

Combining multiple operations with the pipe

We want to explore the relationship between the distance and average delay for each location

- group flights by destination
- summarise to compute distance, average delay, and number of flights
- filter to remove noisy points and Honolulu airports, which is almost twise as fat away as the next closest airport

Combining multiple operations with the pipe

```
by_dest <- group_by(flights, dest)
delay <- summarise(by_dest,
  count = n(),
  dist = mean(distance, na.rm = TRUE),
  delay = mean(arr_delay, na.rm = TRUE)
)
delay <- filter(delay, count > 20, dest != "HNL")
```

Combining multiple operations with the pipe

```
 1  flights %>%
 2    group_by(dest) %>%
 3    summarise(
 4      count = n(),
 5      dist = mean(distance, na.rm = TRUE),
 6      delay = mean(arr_delay, na.rm = TRUE)
 7    ) %>%
 8    filter(count > 20, dest != "HNL")
```

```
# A tibble: 96 × 4
   dest  count  dist delay
   <chr> <int> <dbl> <dbl>
 1 ABQ     254 1826   4.38
 2 ACK     265  199   4.85
 3 ALB     439  143  14.4
 4 ATL   17215  757. 11.3
 5 AUS    2439 1514.  6.02
 6 AVL     275  584.  8.00
 7 BDL     443  116   7.05
 8 BGR     375  378   8.03
 9 BHM     297  866. 16.9
```

```
10 BNA    6333  758. 11.8
# … with 86 more rows
```

Final exercise

Let's look at how the average performance of batters in baseball is related to the number of times they're at bat.

- Use Lahman package to compute the batting average of every major league baseball player
- create batting average variable `ba` (sum of hits(H)/sum of opportunities to his the ball (AB))
- create total opportunities to hit the ball variable `ab`(see above for definition)
- only keep batters with 100 and more opportunities

Final exercise

```
 1  library(Lahman)
 2  batting<-Lahman::Batting
 3  batting %>%
 4    group_by(playerID) %>%
 5    summarise(
 6      ba = sum(H, na.rm = TRUE) / sum(AB, na.rm = TRUE),
 7      ab = sum(AB, na.rm = TRUE)
 8    )%>%filter(ab>100)
```

```
# A tibble: 9,170 × 3
   playerID     ba    ab
   <chr>     <dbl> <int>
 1 aaronha01 0.305 12364
 2 aaronto01 0.229   944
 3 abbated01 0.254  3044
 4 abbeybe01 0.169   225
 5 abbeych01 0.281  1756
 6 abbotfr01 0.209   513
 7 abbotje01 0.263   596
 8 abbotku01 0.256  2044
 9 abercre01 0.223   386
10 abernbr01 0.244   868
# … with 9,160 more rows
```