Statistical Computing with R
Masters in Data Science 503 (S9)
Fourth Batch, SMS, TU, 2025

Shital Bhandary
Associate Professor
Statistics/Bio-statistics, Demography and Public Health Informatics
Patan Academy of Health Sciences, Lalitpur, Nepal
Faculty, Masters in Medical Research, NHRC/Kathmandu University
Faculty, FAIMER Fellowship in Health Professions Education, India/USA

Review Preview (Unit 2, Part 2 & 3)

- Data wrangling
- Data munching
- Tidy data
- “dplyr” package and its use for data manipulation

- Reading database in R
- Big data in R
- Text Mining

Transform/manipulate data with “dplyr”

- To learn five key “dplyr” package functions that allow you to solve the vast majority of your data manipulation challenges:

  - 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.

Data manipulation with “dplyr”

- These six functions provide the verbs for a language of data manipulation.

- All verbs work similarly:

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

- Together these properties make it easy to chain together multiple simple steps to achieve a complex result.

Let’s use them with nycflighst13 data

- library(dplyr)
- library(nycflights13)
- flights
- #> # A tibble: 336,776 × 19
- #> year month day dep_time sched_dep…[1] dep_d…[2] arr_t…[3] sched…[4] arr_d…[5] carrier
- #> <int> <int> <int> <int> <int> <dbl> <int> <int> <dbl> <chr>
- #> 1 2013 1 1 517 515 2 830 819 11 UA
- #> 2 2013 1 1 533 529 4 850 830 20 UA
- #> 3 2013 1 1 542 540 2 923 850 33 AA
- #> 4 2013 1 1 544 545 -1 1004 1022 -18 B6
- #> 5 2013 1 1 554 600 -6 812 837 -25 DL
- #> 6 2013 1 1 554 558 -4 740 728 12 UA
- #> # … with 336,770 more rows, 9 more variables

Filter: What will happen?

- **filter(flights, month == 1, day == 1)**
- #> # A tibble: 842 × 19
- #>   year month   day dep_time sched_dep…[1] dep_d…[2] arr_t…[3] sched…[4] arr_d…[5] carrier
- #>   <int> <int> <int>   <int>   <int>   <dbl>   <int>   <int>   <dbl> <chr>
- #> 1 2013   1   1   517   515   2   830   819   11 UA
- #> 2 2013   1   1   533   529   4   850   830   20 UA
- #> 3 2013   1   1   542   540   2   923   850   33 AA
- #> 4 2013   1   1   544   545   -1   1004   1022   -18 B6
- #> 5 2013   1   1   554   600   -6   812   837   -25 DL
- #> 6 2013   1   1   554   558   -4   740   728   12 UA
- #> # … with 836 more rows, 9 more variables

Are these better?

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

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

- filter(flights, month = 1)   #Why error?
- filter(flights, month == 1)   #Works now? Why?

More with filter:

- filter(flights, month == 11 | month == 12) #What?
- filter(flights, month == 11 | 12) #Works?
- nov_dec <- filter(flights, month %in% c(11, 12)) #Works?

- De Morgan’s Law:
- filter(flights, !(arr_delay > 120 | dep_delay > 120)) #Works?
- filter(flights, arr_delay <= 120, dep_delay <= 120) #Works?

Arrange: Example

- arrange(flights, year, month, day)
- #> # A tibble: 336,776 × 19
- #>   year month   day dep_time sched_dep…[1] dep_d…[2] arr_t…[3] sched…[4] arr_d…[5] carrier
- #>   <int> <int> <int>   <int>   <int>   <dbl>   <int>   <int>   <dbl> <chr>
- #> 1 2013   1   1   517   515   2   830   819   11 UA
- #> 2 2013   1   1   533   529   4   850   830   20 UA
- #> 3 2013   1   1   542   540   2   923   850   33 AA
- #> 4 2013   1   1   544   545   -1   1004   1022   -18 B6
- #> 5 2013   1   1   554   600   -6   812   837   -25 DL
- #> 6 2013   1   1   554   558   -4   740   728   12 UA
- #> # … with 336,770 more rows, 9 more variables

What will happen now?

- Arrange will sort the data in ascending order
- arrange(flights, desc(dep_delay))
- Use desc() to re-order by a column in descending order
- Missing values are always sorted at the end

Select: Example

- # Select columns by name
- 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
- #> # … with 336,770 more rows

- # Select all columns between year and day (inclusive)
- 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
- #> # … with 336,770 more rows

Select: “except” example

- # Select all columns except those from year to day (inclusive)
- select(flights, -(year:day))
- #> # A tibble: 336,776 × 16
- #>   dep_time sched…[1] dep_d…[2] arr_t…[3] sched…[4] arr_d…[5] carrier flight tailnum origin
- #>   <int>   <int>   <dbl>   <int>   <int>   <dbl> <chr>   <int> <chr>   <chr>
- #> 1   517   515   2   830   819   11 UA   1545 N14228 EWR
- #> 2   533   529   4   850   830   20 UA   1714 N24211 LGA
- #> 3   542   540   2   923   850   33 AA   1141 N619AA JFK
- #> 4   544   545   -1   1004   1022   -18 B6   725 N804JB JFK
- #> 5   554   600   -6   812   837   -25 DL   461 N668DN LGA
- #> 6   554   558   -4   740   728   12 UA   1696 N39463 EWR
- #> # … with 336,770 more rows, 6 more variables

Select: More

- There are a number of helper functions you 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”.
- matches("(.)\\1"): selects variables that match a **regular expression**.
- This one matches any variables that contain repeated characters.
- **num_range**("x", 1:3): matches x1, x2 and x3.
- See ?select for more details.

More on regular expression are available here: https://cran.r-project.org/web/packages/stringr/vignettes/regular-expressions.html

Note:

- select() can be used to rename variables, **but it's rarely useful because it drops all of the variables not explicitly mentioned**.
- Instead, **use rename()**, which is a variant of select() that keeps all the variables that aren't explicitly mentioned
- rename(flights, tail_num = tailnum)
- Another option is to use select() in conjunction with the everything() helper.
- This is useful if you have a handful of variables you'd like to move to the start of the data frame.
- select(flights, time_hour, air_time, everything())

Mutate: Example

- Besides selecting sets of existing columns, it's often useful to add new columns that are functions of existing columns.

- That's the job of mutate().

- mutate() always adds new columns at the end of your dataset so we'll start by creating a narrower dataset so we can see the new variables.

```
#Addiing variables in flights_sml:
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
)
```

Mutate: Example

- Besides selecting sets of existing columns, it’s often useful to add new columns that are functions of existing columns.
- That’s the job of mutate().
- mutate() always adds new columns at the end of your dataset so we’ll start by creating a narrower dataset so we can see the new variables.

```
#Adding one more variable:

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

#Note that you/we can refer to columns that you’ve just created
```

Transmute and other useful creation functions More@ https://r4ds.had.co.nz/transform.html

- 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

)

- Arithmetic operators: +, -, *, /, ^
- Modular arithmetic: %/% (integer division) and %% (remainder)
- Use: Compute hour and minute from dep_time with:
- transmute(flights,

  dep_time,

  hour = dep_time %/% 100,

  minute = dep_time %% 100)

Summarise: Works best for group summaries

- summarise(flights, delay = mean(dep_delay, na.rm = TRUE))
- #> # A tibble: 1 × 1
- #>   delay
- #>   <dbl>
- #> 1 12.6

- by_day <- group_by(flights, year, month, day)
- 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
- #> # … with 359 more rows

Multiple operations: pipes

```
#What will happen?
delays <- flights %>%
  group_by(dest) %>%
  summarise(
    count = n(),
    dist = mean(distance, na.rm =
TRUE),
    delay = mean(arr_delay, na.rm =
TRUE)
  ) %>%
  filter(count > 20, dest != "HNL")
```

```
#What will happen?
flights %>%
  group_by(year, month, day) %>%
  summarise(mean =
mean(dep_delay))
```

```
#And now?
flights %>%
  group_by(year, month, day) %>%
  summarise(mean =
mean(dep_delay, na.rm = TRUE))
```

How to remove cancelled flights? And, get summaries by groups!

```
not_cancelled <- flights %>%
          filter(!is.na(dep_delay),
          !is.na(arr_delay))

not_cancelled %>%
  group_by(year, month, day) %>%
  summarise(mean =
mean(dep_delay))
```

- #> # A tibble: 365 × 4
- #> # Groups: year, month [12]
- #>   year month   day mean
- #>   <int> <int> <int> <dbl>
- #> 1 2013   1   1 11.4
- #> 2 2013   1   2 13.7
- #> 3 2013   1   3 10.9
- #> 4 2013   1   4 8.97
- #> 5 2013   1   5 5.73
- #> 6 2013   1   6 7.15
- #> # … with 359 more rows

Counts: Example

- Whenever you do any aggregation, it’s always a good idea to include either a count (n()), or a count of non-missing values (sum(!is.na(x))).
- That way you can check that you’re not drawing conclusions based on very small amounts of data.

```
# What happens now?

delays <- not_cancelled %>%
        group_by(tailnum) %>%
        summarise(
        delay = mean(arr_delay)
)

hist(delays$delay)
```

What happens now?

```
delays <- not_cancelled %>%
	group_by(tailnum) %>%
	summarise(
	delay = mean(arr_delay, na.rm = TRUE),
	n = n()
)
```

```
# Plots
# Can you interpret them?

hist(delays$n)

hist(delays$delay)

plot(delays$n, delays$delay)
```

Useful summary functions: https://r4ds.had.co.nz/transform.html

```
# When do the first and last flights leave each day?
not_cancelled %>%
  group_by(year, month, day) %>%
  summarise(
    first = min(dep_time),
    last = max(dep_time)
  )
```

- # Why is distance to some destinations more variable than to others?

```
not_cancelled %>%
        group_by(dest) %>%
        summarise(distance_sd =
         sd(distance)) %>%
        arrange(desc(distance_sd))
```

Useful summary functions: https://r4ds.had.co.nz/transform.html

```
# Which destinations have the most carriers?

not_cancelled %>%
  group_by(dest) %>%
  summarise(carriers = n_distinct(carrier)) %>%
  arrange(desc(carriers))
```

- # How many flights left before 5am? (these usually indicate delayed flights from the previous day)

```
not_cancelled %>%
  group_by(year, month, day) %>%
  summarise(n_early = sum(dep_time < 500))
```

Useful summary functions: https://r4ds.had.co.nz/transform.html

What proportion of flights are delayed by more than an hour?

```
not_cancelled %>%
  group_by(year, month, day) %>%
  summarise(hour_prop = mean(arr_delay > 60))
```

#Find all groups bigger than a threshold:

```
popular_dests <- flights %>%
  group_by(dest) %>%
  filter(n() > 365)
popular_dests
```

Popular destination: head and tail
(Are these results VALID?)

- head(popular_dests$dest)
- [1] "IAH" "IAH" "MIA" "BQN" "ATL" "ORD“

- IAH = Texas
- MIA = Miami
- BQN = Puerto Rico
- ATL = Atalanta
- ORD = Chichago

- tail(popular_dests$dest)
- [1] "BNA" "DCA" "SYR" "BNA" "CLE" "RDU“

- BNA = Nashville
- DCA = Washigton (Reagan Nat.)
- SYR = New York (Syracuse)
- CLE = Cleveland
- RDU = North Carolina

Bonus: dplyr “slice” function with examples https://dplyr.tidyverse.org/reference/slice.html

#What will happen?

flights %>% slice(1L)

flights %>% slice(n())

flights %>% slice(5:n())

slice(flights,-(1:4))

- flights %>% slice_sample(n=5)
- flights %>% slice_sample(n=5, replace = TRUE)
- **set seed(123)**
- train_data <- flights %>% slice_sample(prop=0.8)
- train_data
- test_data <- flights %>% slice_sample(prop=0.2)
- test_data

Resources for the Next class

- R and Relational database: Chapter 13, R for Data Science, First Edition https://r4ds.had.co.nz/relational-data.html

- R for Data Science, 2nd Edition, Chapter 22: Databases https://r4ds.hadley.nz/databases

- R and Big Data: https://rviews.rstudio.com/2019/07/17/3-big-data-strategies-for-r/

Question/Queries?

Thank you!

@shitalbhandary