R Bookclub

Chapters 2 & 3

Shu Asai

04/03/2019

Chapter 2: Workflow basics

R as a calculator

```r
2+4
```

```
[1] 6
```

```r
pi
```

```
[1] 3.141593
```

```r
(pi+1)/2
```

```
[1] 2.070796
```

Objects & Naming

Use '<-' to attribute value to objects

```r
floors_harwick <- 8
```

```r
floors_gonda <- 19
```

Exercise:

```r
floor_diff <- floors_gonda - floors_harwick
```

```r
floor_diff
```

```
[1] 11
```

Functions

Provide input arguments, Obtain output

Example: **seq()**

- Two arguments needed: seq(from, to)
- R produces a **seq**uence of integers from provided arguments

```
seq(10,20)
```

```
 [1] 10 11 12 13 14 15 16 17 18 19 20
```

Function documentation is found by typing "?" + function name in the Console:

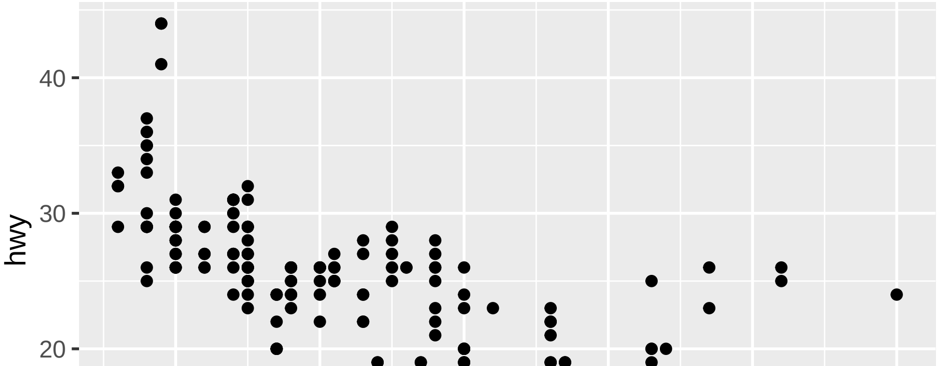
```
?seq
```

Exercises

Find the error in my code

```
ggplot(dota = mpg) +
  geom_point(mapping = aes(x=displ, y=hwy))
```

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x=displ, y=hwy))
```



Exercises cont.

```r
fliter(mpg, cyl=8)
```

```r
filter(mpg, cyl==8)
```

```
# A tibble: 70 x 11
   manufacturer model displ  year   cyl trans drv     cty   hwy fl    class
   <chr>        <chr> <dbl> <int> <int> <chr> <chr> <int> <int> <chr> <chr>
 1 audi         a6 q~   4.2  2008     8 auto~ 4        16    23 p     mids~
 2 chevrolet    c150~   5.3  2008     8 auto~ r        14    20 r     suv
 3 chevrolet    c150~   5.3  2008     8 auto~ r        11    15 e     suv
 4 chevrolet    c150~   5.3  2008     8 auto~ r        14    20 r     suv
 5 chevrolet    c150~   5.7  1999     8 auto~ r        13    17 r     suv
 6 chevrolet    c150~   6     2008     8 auto~ r        12    17 r     suv
 7 chevrolet    corv~   5.7  1999     8 manu~ r        16    26 p     2sea~
 8 chevrolet    corv~   5.7  1999     8 auto~ r        15    23 p     2sea~
 9 chevrolet    corv~   6.2  2008     8 manu~ r        16    26 p     2sea~
10 chevrolet    corv~   6.2  2008     8 auto~ r        15    25 p     2sea~
# ... with 60 more rows
```

Exercises cont.

```r
filter(diamond, carat>3)
```

```r
filter(diamonds, carat>3)
```

```
# A tibble: 32 x 10
   carat cut     color clarity depth table price     x     y     z
   <dbl> <ord>   <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
 1  3.01 Premium I     I1       62.7    58  8040  9.1   8.97  5.67
 2  3.11 Fair    J     I1       65.9    57  9823  9.15  9.02  5.98
 3  3.01 Premium F     I1       62.2    56  9925  9.24  9.13  5.73
 4  3.05 Premium E     I1       60.9    58 10453  9.26  9.25  5.66
 5  3.02 Fair    I     I1       65.2    56 10577  9.11  9.02  5.91
 6  3.01 Fair    H     I1       56.1    62 10761  9.54  9.38  5.31
 7  3.65 Fair    H     I1       67.1    53 11668  9.53  9.48  6.38
 8  3.24 Premium H     I1       62.1    58 12300  9.44  9.4   5.85
 9  3.22 Ideal   I     I1       62.6    55 12545  9.49  9.42  5.92
10  3.5  Ideal   H     I1       62.8    57 12587  9.65  9.59  6.03
# ... with 22 more rows
```

Keyboard Tips & Shortcuts

- Snake Case (Examples: “floors_harwick” , “cohort_A” , “scatter_MCHS”)

```
* Descriptive and readable object names
* Avoid camel case and periods
```

Keyboard Tips & Shortcuts

- Snake Case (Examples: “floors_harwick” , “cohort_A” , “scatter_MCHS”)

```
* Descriptive and readable object names
* Avoid camel case and periods
```

- Use **(Alt) (-)** as a keyboard shortcut for <-

Keyboard Tips & Shortcuts

- Snake Case (Examples: “floors_harwick” , “cohort_A” , “scatter_MCHS”)

```
* Descriptive and readable object names
* Avoid camel case and periods
```

- Use **(Alt) (-)** as a keyboard shortcut for <-
- Using “Tab” to explore objects

Keyboard Tips & Shortcuts

- Snake Case (Examples: “floors_harwick” , “cohort_A” , “scatter_MCHS”)

```
* Descriptive and readable object names
* Avoid camel case and periods
```

- Use **(Alt) (-)** as a keyboard shortcut for <-
- Using “Tab” to explore objects
- RStudio Environment & History: (Cmd) (Ctrl) (-) (up arrow)

Keyboard Tips & Shortcuts

- Snake Case (Examples: “floors_harwick” , “cohort_A” , “scatter_MCHS”)

```
* Descriptive and readable object names
* Avoid camel case and periods
```

- Use **(Alt) (-)** as a keyboard shortcut for <-
- Using “Tab” to explore objects
- RStudio Environment & History: (Cmd) (Ctrl) (-) (up arrow)
- Shortcut Cheatsheet: (Alt) (Shift) (K)

Chapter 3: Transformations with dplyr

Variable Types

- *int* (Integers)
- *dbl* (Doubles or real numbers)
- *chr* (Character vectors or strings)
- *dttm* (Date / Times)

?dplyr

What is it?

`dplyr is an R package containing functions useful to transform and manipul`

Why dplyr?

`Intuitive, readable, and fast!`

Five key dplyr functions:

1. `filter`
2. `arrange`
3. `select`
4. `mutate`
5. `summarize`

dplyr Function Workflow

Load in the dplyr package

```
library(dplyr)
```

1. Provide the function a dataframe

dplyr Function Workflow

Load in the dplyr package

```
library(dplyr)
```

1. Provide the function a dataframe
2. Describe what to do to the dataframe with column specifications

dplyr Function Workflow

Load in the dplyr package

```r
library(dplyr)
```

1. Provide the function a dataframe
2. Describe what to do to the dataframe with column specifications
3. Output to a new dataframe

The Data

```
flights
```

```
# A tibble: 336,776 x 19
    year month   day dep_time sched_dep_time dep_delay arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>
 1  2013     1     1      517            515         2      830
 2  2013     1     1      533            529         4      850
 3  2013     1     1      542            540         2      923
 4  2013     1     1      544            545        -1     1004
 5  2013     1     1      554            600        -6      812
 6  2013     1     1      554            558        -4      740
 7  2013     1     1      555            600        -5      913
 8  2013     1     1      557            600        -3      709
 9  2013     1     1      557            600        -3      838
10  2013     1     1      558            600        -2      753
# ... with 336,766 more rows, and 12 more variables: sched_arr_time <int>,
#   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
```

‘filter’: Subset Rows

'filter' workflow

- Provide the function a dataframe

```r
filter(flights)
```

- Describe what to do to the dataframe with column specifications

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

- Output to a new dataframe

```r
(flights_1_1 <- filter(flights,  month == 1 , day == 1))
```

```
# A tibble: 842 x 19
   year month   day dep_time sched_dep_time dep_delay arr_time
  <int> <int> <int>    <int>          <int>     <dbl>    <int>
1  2013     1     1      517            515         2      830
2  2013     1     1      533            529         4      850
3  2013     1     1      542            540         2      923
4  2013     1     1      544            545        -1     1004
```

Useful Operators

Use operators to specify your "filter" statement.

1. "and" statement: **&**

Find all flights operated by United ("UA"), American ("AA"), or Delta ("DL")

```
filter(flights, carrier %in% c( "UA", "AA",  "DL" ))
```

Useful Operators

Use operators to specify your "filter" statement.

1. "and" statement: **&**
2. "or" statement: |

Find all flights operated by United ("UA"), American ("AA"), or Delta ("DL")

```
filter(flights, carrier %in% c( "UA", "AA",  "DL" ))
```

Useful Operators

Use operators to specify your "filter" statement.

1. "and" statement: **&**
2. "or" statement: |
3. "not" statement: **!=**

Find all flights operated by United ("UA"), American ("AA"), or Delta ("DL")

```
filter(flights, carrier %in% c( "UA", "AA",  "DL" ))
```

Useful Operators

Use operators to specify your "filter" statement.

1. "and" statement: **&**
2. "or" statement: |
3. "not" statement: **!=**
4. extended "or" statement: **%in%**

Find all flights operated by United ("UA"), American ("AA"), or Delta ("DL")

```
filter(flights, carrier %in% c( "UA", "AA",  "DL" ))
```

More Exercises

Find flights that had an arrival delay of two or more hours:

```
filter(flights, arr_delay >=2)
```

Find flights that flew to “IAH” or “HOU”

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

Find flights that were delayed by at least an hour, but made up over 30 minutes.

```
filter(flights, dep_delay >= 1 & arr_delay < -30)
```

Exercises cont.

The *between(column, min, max)* function is also a useful dplyr verb.

```
filter( dataframe, between(column, min, max) )
```

How can we use it in this example?

Find flights that departed between July, August, and September.

```
filter(flights, between(month,7,9))
```

```
# A tibble: 3 x 19
   year month   day dep_time sched_dep_time dep_delay arr_time
  <int> <int> <int>    <int>          <int>     <dbl>    <int>
1  2013     7     1        1           2029       212      236
2  2013     7     1        2           2359         3      344
3  2013     7     1       29           2245       104      151
# ... with 12 more variables: sched_arr_time <int>, arr_delay <dbl>,
#   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
#   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
```

Exercises cont.

Which flights are missing a “dep_time”?

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

```
# A tibble: 3 x 19
   year month   day dep_time sched_dep_time dep_delay arr_time
  <int> <int> <int>    <int>          <int>     <dbl>    <int>
1  2013     1     1       NA           1630        NA       NA
2  2013     1     1       NA           1935        NA       NA
3  2013     1     1       NA           1500        NA       NA
# ... with 12 more variables: sched_arr_time <int>, arr_delay <dbl>,
#   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
#   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
#   time_hour <dttm>
```

What could these rows represent?

‘arrange’: Order your Data

'arrange'

Order dataset by provided columns in ascending order

```r
(flight_month <- arrange(flights, month))
```

```
# A tibble: 336,776 x 19
    year month   day dep_time sched_dep_time dep_delay arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>
 1  2013     1     1      517            515         2      830
 2  2013     1     1      533            529         4      850
 3  2013     1     1      542            540         2      923
 4  2013     1     1      544            545        -1     1004
 5  2013     1     1      554            600        -6      812
 6  2013     1     1      554            558        -4      740
 7  2013     1     1      555            600        -5      913
 8  2013     1     1      557            600        -3      709
 9  2013     1     1      557            600        -3      838
10  2013     1     1      558            600        -2      753
# ... with 336,766 more rows, and 12 more variables: sched_arr_time <int>,
```

‘arrange’ cont.

Arranging the data in the opposite direction (descending order):

```
(flight_year_month <- arrange(flights, desc(month)))
```

```
# A tibble: 336,776 x 19
    year month   day dep_time sched_dep_time dep_delay arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>
 1  2013    12     1       13           2359        14      446
 2  2013    12     1       17           2359        18      443
 3  2013    12     1      453            500        -7      636
 4  2013    12     1      520            515         5      749
 5  2013    12     1      536            540        -4      845
 6  2013    12     1      540            550       -10     1005
 7  2013    12     1      541            545        -4      734
 8  2013    12     1      546            545         1      826
 9  2013    12     1      549            600       -11      648
10  2013    12     1      550            600       -10      825
# ... with 336,766 more rows, and 12 more variables: sched_arr_time <int>,
```

‘arrange’: cont.

Multiple arguments in the arrange statement.

```
flight_year_dep <- arrange(flights, year, dep_time)
```

```
# A tibble: 4 x 19
   year month   day dep_time sched_dep_time dep_delay arr_time
  <int> <int> <int>    <int>          <int>     <dbl>    <int>
1  2013     1    13        1           2249        72      108
2  2013     1    31        1           2100       181      124
3  2013    11    13        1           2359         2      442
4  2013    12    16        1           2359         2      447
# ... with 12 more variables: sched_arr_time <int>, arr_delay <dbl>,
#   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
#   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
#   time_hour <dttm>
```

What happens to NA values in arrange() statements?

Handling NAs and Missing Values

To determine if a value is missing/NA, use the 'is.na()' function:

```
x <- NA
is.na(x)
```

```
[1] TRUE
```

The following result in NA:

```
NA > 5
10 == NA
NA + 10
NA / 2
```

Rule- if the calculation involves NA, result will almost always be NA.

NAs: Exceptions

```r
NA ^ 0
```

```
[1] 1
```

```r
NA | TRUE
```

```
[1] TRUE
```

Exercises

How can we sort missing values to the top of the dataset?

```
arrange(flights, desc(is.na(dep_time)))
```

Find flights that were most delayed, then sort by those that left earliest

```
arrange(flights, desc(dep_delayed), departure)
```

Find the longest flights

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

‘select’: Select Columns of Data

‘select’

```r
flights_ymd <- select(flights, year, month, day)
flights_ymd
```

```
# 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’ cont.

Selecting all columns between ‘carrier’ and ‘origin’

```
select(flights, carrier:origin)
```

```
# A tibble: 336,776 x 4
   carrier flight tailnum origin
   <chr>    <int> <chr>   <chr>
 1 UA        1545 N14228  EWR
 2 UA        1714 N24211  LGA
 3 AA        1141 N619AA  JFK
 4 B6         725 N804JB  JFK
 5 DL         461 N668DN  LGA
 6 UA        1696 N39463  EWR
 7 B6         507 N516JB  EWR
 8 EV        5708 N829AS  LGA
 9 B6          79 N593JB  JFK
10 AA         301 N3ALAA  LGA
# ... with 336,766 more rows
```

‘select’ cont.

Selecting all columns except those from ‘carrier’ to ‘origin’

```r
select(flights, -(carrier : origin))
```

```
# A tibble: 336,776 x 15
    year month   day dep_time sched_dep_time dep_delay arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>
 1  2013     1     1      517            515         2      830
 2  2013     1     1      533            529         4      850
 3  2013     1     1      542            540         2      923
 4  2013     1     1      544            545        -1     1004
 5  2013     1     1      554            600        -6      812
 6  2013     1     1      554            558        -4      740
 7  2013     1     1      555            600        -5      913
 8  2013     1     1      557            600        -3      709
 9  2013     1     1      557            600        -3      838
10  2013     1     1      558            600        -2      753
# ... with 336,766 more rows, and 8 more variables: sched_arr_time <int>,
```

‘select’ helper statements

starts_with(“x”)

`selects columns that begin with "x"`

ends_with(“type”)

`selects columns that end with "type"`

contains(“gene”)

`selects columns that contain "gene"`

num_range(“x”, 1:3)

`selects columns titled, "x1", "x2", and "x3"`

Exercises

What happens if you refer to the same column multiple times in a ‘select’ call?

```r
select(flights, year, year)
```

```
# A tibble: 3 x 1
   year
  <int>
1  2013
2  2013
3  2013
```

Exercises cont.

What does the 'one_of' function do? How could it be useful in a select statement?

Given a vector of characters, 'one_of' finds column names that match in the vector.

```
variables <- c("year", "month", "day","arr_time")
```

```
select(flights, one_of(variables))
```

```
# A tibble: 3 x 4
   year month   day arr_time
  <int> <int> <int>    <int>
1  2013     1     1      830
2  2013     1     1      850
3  2013     1     1      923
```

Column Manipulation

Use ‘rename’ to change column names

Syntax: rename(data, *new_column_name* = old_column_name)

```
rename(flights, depart_delay = dep_delay)
```

```
# A tibble: 336,776 x 19
   year month   day dep_time sched_dep_time depart_delay arr_time
  <int> <int> <int>    <int>          <int>        <dbl>    <int>
1  2013     1     1      517            515            2      830
2  2013     1     1      533            529            4      850
3  2013     1     1      542            540            2      923
4  2013     1     1      544            545           -1     1004
5  2013     1     1      554            600           -6      812
6  2013     1     1      554            558           -4      740
7  2013     1     1      555            600           -5      913
8  2013     1     1      557            600           -3      709
9  2013     1     1      557            600           -3      838
```

Column Manipulation cont.

Use 'select' and 'everything' together to rearrange columns

```r
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
   <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>,
```

mutate: Create Columns

‘mutate’

Create and append columns to exisiting data with ‘mutate’

Syntax:

mutate(data, *new_column_name* = column manipulation)

‘mutate’ example

```
flights_A <- select(flights, year : day, arr_delay, starts_with("dep"))
```

```
# A tibble: 3 x 6
   year month   day arr_delay dep_time dep_delay
  <int> <int> <int>     <dbl>    <int>     <dbl>
1  2013     1     1        11      517         2
2  2013     1     1        20      533         4
3  2013     1     1        33      542         2
```

```
mutate(flights_A,
       gain = arr_delay - dep_delay)
```

```
# A tibble: 3 x 7
   year month   day arr_delay dep_time dep_delay  gain
  <int> <int> <int>     <dbl>    <int>     <dbl> <dbl>
1  2013     1     1        11      517         2     9
2  2013     1     1        20      533         4    16
3  2013     1     1        33      542         2    31
```

'mutate' Multiple Columns

```
mutate(flights_A,
       gain = arr_delay - dep_delay,
       arr_dep_diff = arr_delay - dep_time)
```

```
# A tibble: 5 x 8
   year month   day arr_delay dep_time dep_delay  gain arr_dep_diff
  <int> <int> <int>     <dbl>    <int>     <dbl> <dbl>        <dbl>
1  2013     1     1        11      517         2     9         -506
2  2013     1     1        20      533         4    16         -513
3  2013     1     1        33      542         2    31         -509
4  2013     1     1       -18      544        -1   -17         -562
5  2013     1     1       -25      554        -6   -19         -579
```

‘transmute’

‘transmute’ is the same as ‘mutate’, but will instead only keep the new variables:

```
transmute(flights_A,
          gain = arr_delay - dep_delay,
          arr_dep_diff = arr_delay - dep_time)
```

```
# A tibble: 336,776 x 2
    gain arr_dep_diff
   <dbl>        <dbl>
 1     9         -506
 2    16         -513
 3    31         -509
 4   -17         -562
 5   -19         -579
 6    16         -542
 7    24         -536
 8   -11         -571
 9    -5         -565
```

Useful Functions When Mutating

- */ , + , -, * , ^*
- *sum(), mean()*
- *%/% (integer division)*

```r
4 %/% 2
```

```
[1] 2
```

- *%% (remainder)*

```r
4 %% 2
```

```
[1] 0
```

Useful Functions When Mutating cont.

Logs

- Natural logarithms

```
log()
```

- Binary logarithms (base 2)

```
log2()
```

- Common logarithms (base 10)

```
log10()
```

Useful Functions cont : Offsets

'lead' & 'lag'

```
(tn <- 1:10)
```

```
 [1]  1  2  3  4  5  6  7  8  9 10
```

```
(lead(tn))
```

```
 [1]  2  3  4  5  6  7  8  9 10 NA
```

```
(lag(tn))
```

```
 [1] NA  1  2  3  4  5  6  7  8  9
```

Useful Functions cont : Aggregates

Cumulative and rolling calculations

What is the sum of all previous values (including current value)?

```r
cumsum(tn)
```

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

What is the mean of all previous values (including current value)?

```r
cummean(tn)
```

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

others: ‘cummin’ , ‘cummax’, ‘cumprod’

Useful Functions cont : Comparisons

Logical comparisons

```
<
>
<=
>=
!=
==
```

Exercises

Find the 10 most delayed flights using mutate and the 'min_rank' function.

Note: 'min_rank' assigns a number equal to a number of elements less than that value plus one.

```
mr <- c(4, 2, 7, 7, 7 , 0, 10)
min_rank(mr)
```

```
[1] 3 2 4 4 4 1 7
```

```
min_rank(desc(mr))
```

```
[1] 5 6 2 2 2 7 1
```

Exercises cont.

Find the 10 most delayed flights using mutate and the ‘min_rank’ function.

```
flights_delay <- mutate(flights, delay_rank = min_rank(desc(dep_delay)))
```

```
flights_delay <- select(flights_delay, delay_rank, dep_delay, everything()
```

```
arrange(flights_delay, delay_rank)[1:5,]
```

```
# A tibble: 5 x 20
  delay_rank dep_delay  year month   day dep_time sched_dep_time arr_time
       <int>     <dbl> <int> <int> <int>    <int>          <int>    <int>
1          1      1301  2013     1     9      641            900     1242
2          2      1137  2013     6    15     1432           1935     1607
3          3      1126  2013     1    10     1121           1635     1239
4          4      1014  2013     9    20     1139           1845     1457
5          5      1005  2013     7    22      845           1600     1044
# ... with 12 more variables: sched_arr_time <int>, arr_delay <dbl>,
#   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
```

Exercises cont.

‘summarize’: Aggregate Calculations

‘summarize’

Collapses data into a single row:

```
summarize(flights, avg_delay = mean(dep_delay, na.rm = TRUE))
```

```
# A tibble: 1 x 1
  avg_delay
      <dbl>
1      12.6
```

‘summarize’ is not too useful on its own.
It is often used in tandem with

Pipes and ‘group_by’

Pipes and ‘group_by’

Grouped Summaries with “group_by”

Implement *group_by* to use dplyr functions on a *grouped* dataframe

Example:

```
by_day <- group_by(flights, year, month, day)
```

The above code does nothing to the structure of the data. The data is merely coerced into a grouped dataframe.

```
summarize(by_day, delay = mean(dep_delay, na.rm = TRUE))[1:3,]
```

```
# A tibble: 3 x 4
# Groups:   year, month [1]
   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
```

A note on ‘na.rm’

Recall, *most* calculations that include ‘NA’ values will result in an ‘NA’.

```
mean(c(2,3,4,NA, 6))
```

```
[1] NA
```

Omit the ‘NA’ values in the calculation with use the option *na.rm=TRUE*

```
mean(c(2,3,4,NA,6), na.rm = TRUE)
```

```
[1] 3.75
```

Ungrouping

Data can be ungrouped at any time

```
by_day <- group_by(flights, year, month, day)
```

```
orig_data <- ungroup(by_day)
```

Piping with ‘%>%’

Pipes allow multiple ‘dplyr’ functions to be used consecutively.

For example, you may want to take your original data, and then:

1. Select specific columns
2. Filter for observations
3. Create (mutate) another column

Data **%>%** Select columns **%>%** Filter observations **%>%** Create a column

Pipe Framework

```
flight_a <- flights %>% select(year, air_time, dest) %>% mutate(air_time_ho
```

1. Provide a dataset

```
# A tibble: 3 x 4
   year air_time dest  air_time_hours
  <int>    <dbl> <chr>          <dbl>
1  2013      227 IAH             3.78
2  2013      227 IAH             3.78
3  2013      160 MIA             2.67
```

Pipe Framework

```
flight_a <- flights %>% select(year, air_time, dest) %>% mutate(air_time_ho
```

1. Provide a dataset
2. Pipe (%>%)

```
# A tibble: 3 x 4
   year air_time dest  air_time_hours
  <int>    <dbl> <chr>          <dbl>
1  2013      227 IAH             3.78
2  2013      227 IAH             3.78
3  2013      160 MIA             2.67
```

Pipe Framework

```
flight_a <- flights %>% select(year, air_time, dest) %>% mutate(air_time_ho
```

1. Provide a dataset
2. Pipe (%>%)
3. Input desired ‘dplyr’ function

```
# A tibble: 3 x 4
   year air_time dest  air_time_hours
  <int>    <dbl> <chr>          <dbl>
1  2013      227 IAH             3.78
2  2013      227 IAH             3.78
3  2013      160 MIA             2.67
```

Pipe Framework

```
flight_a <- flights %>% select(year, air_time, dest) %>% mutate(air_time_ho
```

1. Provide a dataset
2. Pipe (%>%)
3. Input desired 'dplyr' function
4. Repeat steps 2 - 3 as needed

```
# A tibble: 3 x 4
   year air_time dest  air_time_hours
  <int>    <dbl> <chr>          <dbl>
1  2013      227 IAH             3.78
2  2013      227 IAH             3.78
3  2013      160 MIA             2.67
```

Pipe Framework

```
flight_a <- flights %>% select(year, air_time, dest) %>% mutate(air_time_ho
```

1. Provide a dataset
2. Pipe (%>%)
3. Input desired 'dplyr' function
4. Repeat steps 2 - 3 as needed
5. Output into a new object

```
# A tibble: 3 x 4
   year air_time dest  air_time_hours
  <int>    <dbl> <chr>          <dbl>
1  2013      227 IAH             3.78
2  2013      227 IAH             3.78
3  2013      160 MIA             2.67
```

Pipes and 'group_by' together

Using both pipes and 'group_by' allows data manipulation to groups of data.

Consider the question: What was the average departure delay in each month?

'group_by' month, use the 'summarize' function

```
flights %>% group_by(month) %>% summarize(avg_delay = mean(dep_delay,na.rm
```

```
# A tibble: 4 x 2
  month avg_delay
  <int>     <dbl>
1     1      10.0
2     2      10.8
3     3      13.2
4     4      13.9
```

Useful Summary Functions

```
flights %>% group_by(month) %>% summarize()
```

Location: *mean(x), median(x)*

Spread: *sd(x), IQR(x), mad(x) (median absolute deviation)*

Rank: *min(x), quantile(x, 0.5) , max(x)*

Position: *first(x) , nth(x, 5), last(x)*

Counts: *n(), sum(is.na(x)), sum(!(is.na(x))), n_distinct(x)*

Find the sum of 'NAs' in the origin column in each month:

```
flights %>% group_by(month) %>% summarize(na_origin = sum(is.na(origin)))
```

Exercises

Provide another approach to attain output from the following:

```
not_cancelled %>% count(dest)
```

Essentially: Find all flights that are not cancelled.

One solution:

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

Other solutions?

If a flight never departs, then it won't arrive. But a flight could also depart and not arrive (crashes, lost flights). We could use 'arr_delay' as a proxy to define cancelled flights.

Exercises cont.

For each plane, count the number of flights before the first delay of greater than one hour.

What dplyr verbs do we want to use?

```
flights %>%
  arrange(tailnum, year, month, day) %>%
  group_by(tailnum) %>%
  mutate(delay_gt1hr = dep_delay > 60) %>%
  mutate(before_delay = cumsum(delay_gt1hr)) %>%
  filter(before_delay < 1) %>%
  count(sort = TRUE)
```

Exercises

For each destination, compute the total minutes of delay.

For each flight, compute the proportion of the total delay for its destination.

```
flights %>%
  filter(!is.na(arr_delay), arr_delay > 0) %>%
  group_by(dest) %>%
  mutate(
    arr_delay_total = sum(arr_delay),
    arr_delay_prop = arr_delay / arr_delay_total) %>%
  ungroup()
```

Exercises cont.

Find destinations that are flown by at least two carriers.

```
dest_2carriers <- flights %>%
  # keep only unique carrier,dest pairs
  select(dest, carrier) %>%
  group_by(dest, carrier) %>%
  filter(row_number() == 1) %>%
  # count carriers by destination
  group_by(dest) %>%
  mutate(n_carrier = n_distinct(carrier)) %>%
  filter(n_carrier >= 2)
```