Eddie Aguilar

Filter rows with `filter()`

✓ filter()

`filter()` lets you use a logical test to extract specific rows from a data frame. To use `filter()`, pass it the data frame followed by one or more logical tests. `filter()` will return every row that passes each logical test.

So for example, we can use `filter()` to select every flight in flights that departed on January 1st. Click Run Code to give it a try:

R Code | Start Over | Run Code

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

year <int>	mo... <int>	d... <int>	dep_time <int>	sched_dep_time <int>	dep_delay <dbl>	arr_time <int>	sched_arr_time <int>	arr_delay <dbl>	carrier <chr>
2013	1	1	517	515	2	830	819	11	UA
2013	1	1	533	529	4	850	830	20	UA
2013	1	1	542	540	2	923	850	33	AA
2013	1	1	544	545	-1	1004	1022	-18	B6
2013	1	1	554	600	-6	812	837	-25	DL
2013	1	1	554	558	-4	740	728	12	UA
2013	1	1	555	600	-5	913	854	19	B6
2013	1	1	557	600	-3	709	723	-14	EV
2013	1	1	557	600	-3	838	846	-8	B6
2013	1	1	558	600	-2	753	745	8	AA

1-10 of 842 rows | 1-10 of 19 columns Previous 1 2 3 4 5 6 ... 85 Next

✓ output

Like all dplyr functions, `filter()` returns a new data frame for you to save or use. It doesn't overwrite the old data frame.

If you want to save the output of `filter()`, you'll need to use the assignment operator, `<-`.

Rerun the command in the code chunk below, but first arrange to save the output to an object named `jan1`.

R Code | Start Over | Solution | Run Code

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

year <int>	mo... <int>	d... <int>	dep_time <int>	sched_dep_time <int>	dep_delay <dbl>	arr_time <int>	sched_arr_time <int>	arr_delay <dbl>	carrier <chr>
2013	1	1	517	515	2	830	819	11	UA
2013	1	1	533	529	4	850	830	20	UA
2013	1	1	542	540	2	923	850	33	AA
2013	1	1	544	545	-1	1004	1022	-18	B6
2013	1	1	554	600	-6	812	837	-25	DL
2013	1	1	554	558	-4	740	728	12	UA

Logical Comparisons

Comparison operators

R provides a suite of comparison operators that you can use to compare values: `>`, `>=`, `<`, `<=`, `!=` (not equal), and `==` (equal). Each creates a logical test. For example, is `pi` greater than three?

```
pi > 3
```

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

When you place a logical test inside of `filter()`, filter applies the test to each row in the data frame and then returns the rows that pass, as a new data frame.

Our code above returned every row whose month value was equal to one *and* whose day value was equal to one.

Watch out!

When you start out with R, the easiest mistake to make is to test for equality with `=` instead of `==`. When this happens you'll get an informative error:

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

```
## Error in `filter()`:
## ! We detected a named input.
## ℹ This usually means that you've used `=` instead of `==`.
## ℹ Did you mean `month == 1`?
```

Multiple tests

If you give `filter()` more than one logical test, `filter()` will combine the tests with an implied "and." In other words, `filter()` will return only the rows that return `TRUE` for every test. You can combine tests in other ways with Boolean operators...

Previous Topic Next Topic

✓ Test Your Knowledge

What will the following code return? `filter(flights, month == 11 | month == 12)`

✓ Every flight that departed in November *or* December

✗ Every flight that departed in November *and* December

✗ Every flight *except for* those that departed in November or December

✗ An error. This is an incorrect way to combine tests.

Correct!

Common mistakes

In R, the order of operations doesn't work like English. You can't write `filter(flights, month == 11 | 12)`, even though you might say "finds all flights that departed in November or December". Be sure to write out a *complete* test on each side of a boolean operator.

Here are four more tips to help you use logical tests and Boolean operators in R:

1. A useful short-hand for this problem is `x %in% y`. This will select every row where `x` is one of the values in `y`. We could use it to rewrite the code in the question above:

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

2. Sometimes you can simplify complicated subsetting by remembering De Morgan's law: `!(x & y)` is the same as `!x | !y`, and `!(x | y)` is the same as `!x & !y`. For example, if you 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:

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

3. As well as `&` and `|`, R also has `&&` and `||`. Don't use them with `filter()`! You'll learn when you should use them later.

4. Whenever you start using complicated, multipart expressions in `filter()`, consider making them explicit variables instead. That makes it much easier to check your work. You'll learn how to create new variables shortly.

Previous Topic Next Topic

Missing values

NA

Missing values can make comparisons tricky in R. R uses `NA` to represent missing or unknown values. `NA`s are "contagious" because almost any operation involving an unknown value (`NA`) will also be unknown (`NA`). For example, can you determine what value these expressions that use missing values should evaluate to? Make a prediction and then click "Submit Answer".

R Code | Start Over | Run Code | Submit Answer

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

```
[1] NA
```

```
[1] NA
```

```
[1] NA
```

```
[1] NA
```

```
[1] NA
```

is.na()

The most confusing result above is this one:

```
NA == NA
```

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

It's easiest to understand why this is true with a bit more context:

```
# Let x be Mary's age. We don't know how old she is.
x <- NA

# Let y be John's age. We don't know how old he is.
y <- NA

# Are John and Mary the same age?
x == y
```

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

```
# We don't know!
```

filter() and NAs

`filter()` only includes rows where the condition is `TRUE`; it excludes both `FALSE` and `NA` values. If you want to preserve missing values, ask for them explicitly:

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

x
<dbl>
3

1 row

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

x
<dbl>
NA
3

2 rows

Use the code chunks below to find all flights that

1. Had an arrival delay of two or more hours

R Code | Start Over | Solution | Run Code

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

carrier <chr>	flight <int>	tailnum <chr>	origin <chr>	dest <chr>	air_time <dbl>	distance <dbl>	hour <dbl>	minute <dbl>	time_hour <dttm>
MQ	4576	N531MQ	LGA	CLT	118	544	6	30	2013-01-01 06:00:00
MQ	3944	N942MQ	JFK	BWI	41	184	18	35	2013-01-01 18:00:00
UA	856	N534UA	EWR	BOS	37	200	7	33	2013-01-01 07:00:00
UA	1086	N76502	LGA	IAH	248	1416	9	0	2013-01-01 09:00:00
EV	4497	N17984	EWR	RIC	63	277	13	10	2013-01-01 13:00:00
B6	525	N231JB	EWR	MCO	152	937	13	40	2013-01-01 13:00:00
EV	4181	N21197	EWR	MCI	234	1092	14	45	2013-01-01 14:00:00
EV	5712	N826AS	JFK	IAD	53	228	13	59	2013-01-01 13:00:00
EV	4092	N16911	EWR	DAY	119	533	16	30	2013-01-01 16:00:00
MQ	4622	N504MQ	LGA	BNA	154	764	16	20	2013-01-01 16:00:00

1-10 of 1,000 rows | 10-19 of 19 columns Previous 1 2 3 4 5 6 … 100 Next

2. Flew to Houston (`IAH` or `HOU`)

R Code | Start Over | Hint | Run Code

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

year <int>	mo... <int>	d... <int>	dep_time <int>	sched_dep_time <int>	dep_delay <dbl>	arr_time <int>	sched_arr_time <int>	arr_delay <dbl>	carrier <chr>
2013	10	10	631	630	1	902	919	-17	UA
2013	10	10	720	725	-5	944	1008	-24	UA
2013	10	10	738	720	18	1001	1000	1	UA
2013	10	10	738	740	-2	1015	1035	-20	WN
2013	10	10	818	825	-7	1123	1115	8	B6
2013	10	10	850	855	-5	1139	1156	-17	UA
2013	10	10	858	902	-4	1154	1148	6	UA
2013	10	10	1034	1030	4	1328	1326	2	UA
2013	10	10	1036	1029	7	1303	1318	-15	UA
2013	10	10	1148	1156	-8	1405	1438	-33	UA

961-970 of 1,000 rows | 1-10 of 19 columns Previous 1 … 95 96 97 98 99 100 Next

3. Were operated by United (`UA`), American (`AA`), or Delta (`DL`)

R Code | Start Over | Hint | Run Code

```
1
2 filter(flights, carrier %in% c("UA", "AA", "DL"))
3
```

year <int>	mo... <int>	d... <int>	dep_time <int>	sched_dep_time <int>	dep_delay <dbl>	arr_time <int>	sched_arr_time <int>	arr_delay <dbl>	carrier <chr>
2013	1	1	517	515	2	830	819	11	UA
2013	1	1	533	529	4	850	830	20	UA
2013	1	1	542	540	2	923	850	33	AA
2013	1	1	554	600	-6	812	837	-25	DL
2013	1	1	554	558	-4	740	728	12	UA
2013	1	1	558	600	-2	753	745	8	AA
2013	1	1	558	600	-2	924	917	7	UA
2013	1	1	558	600	-2	923	937	-14	UA
2013	1	1	559	600	-1	941	910	31	AA
2013	1	1	559	600	-1	854	902	-8	UA

1-10 of 1,000 rows | 1-10 of 19 columns Previous 1 2 3 4 5 6 ... 100 Next

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

R Code | Start Over | Hint | Run Code

```
1 filter(flights, month %in% 7:9)
2
3
```

year <int>	mo... <int>	d... <int>	dep_time <int>	sched_dep_time <int>	dep_delay <dbl>	arr_time <int>	sched_arr_time <int>	arr_delay <dbl>	carrier <chr>
2013	7	1	1	2029	212	236	2359	157	B6
2013	7	1	2	2359	3	344	344	0	B6
2013	7	1	29	2245	104	151	1	110	B6
2013	7	1	43	2130	193	322	14	188	B6
2013	7	1	44	2150	174	300	100	120	AA
2013	7	1	46	2051	235	304	2358	186	B6
2013	7	1	48	2001	287	308	2305	243	VX
2013	7	1	58	2155	183	335	43	172	B6
2013	7	1	100	2146	194	327	30	177	B6
2013	7	1	100	2245	135	337	135	122	B6

1-10 of 1,000 rows | 1-10 of 19 columns Previous 1 2 3 4 5 6 ... 100 Next

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

R Code | Start Over | Hint | Run Code

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

year <int>	mo... <int>	d... <int>	dep_time <int>	sched_dep_time <int>	dep_delay <dbl>	arr_time <int>	sched_arr_time <int>	arr_delay <dbl>	carrier <chr>
2013	10	7	1350	1350	0	1736	1526	130	EV
2013	5	23	1810	1810	0	2208	2000	128	MQ
2013	7	1	905	905	0	1443	1223	140	DL

3 rows | 1-10 of 19 columns

6. Were delayed more than an hour, but made up more than 30 minutes in flight

R Code | Start Over | Hint | Run Code

```
filter(flights, dep_delay > 60  & dep_delay - arr_delay > 30)
```

year <int>	month <int>	day <int>	dep_time <int>	sched_dep_time <int>	dep_delay <dbl>
2013	1	1	2205	1720	285
2013	1	1	2326	2130	116
2013	1	3	1503	1221	162
2013	1	3	1839	1700	99
2013	1	3	1850	1745	65
2013	1	3	1941	1759	102
2013	1	3	1950	1845	65
2013	1	3	2257	2000	177
2013	1	4	1917	1700	137
2013	1	4	2010	1745	145

1-10 of 1,000 rows | 1-6 of 19 columns

Previous 1 2 3 4 5 6 ... 100 Next

7. Departed between midnight and 6am (inclusive)

R Code | Start Over | Hint | Run Code

```
filter(flights, dep_time > 0 & dep_time < 600)  %>%
arrange(dep_time)
```

year <int>	month <int>	day <int>	dep_time <int>	sched_dep_time <int>	dep_delay <dbl>
2013	4	19	121	1940	341
2013	4	22	121	2245	156
2013	6	28	121	1659	502
2013	7	20	121	2359	82
2013	2	23	122	2359	83
2013	2	23	122	2059	263
2013	2	27	122	2250	152
2013	2	27	122	2358	84
2013	3	18	122	2245	157
2013	1	25	123	2000	323

991-1000 of 1,000 rows | 1-6 of 19 columns Previous 1 … 95 96 97 98 99 100 Next

✓ Exercise 2

Another useful dplyr filtering helper is `between()`. What does it do? Can you use `between()` to simplify the code needed to answer the previous challenges?

R Code | Start Over | Run Code

```
filter(flights, between(dep_time, 1, 559))  %>%
arrange(dep_time)
```

year <int>	month <int>	day <int>	dep_time <int>	sched_dep_time <int>	dep_delay <dbl>	arr_time <int>
2013	4	19	121	1940	341	314
2013	4	22	121	2245	156	230
2013	6	28	121	1659	502	329
2013	7	20	121	2359	82	506
2013	2	23	122	2359	83	554
2013	2	23	122	2059	263	345
2013	2	27	122	2250	152	227
2013	2	27	122	2358	84	600
2013	3	18	122	2245	157	226

✓ Exercise 3

How many flights have a missing `dep_time`? What other variables are missing? What might these rows represent?

R Code | Start Over | Hint | Run Code | Submit Answer

```
filter(flights, is.na(dep_time))  %>%
count()
```

n
<int>
8255

1 row

✓ Exercise 4

Why is `NA ^ 0` not missing? Why is `NA | TRUE` not missing? Why is `FALSE & NA` not missing? Can you figure out the general rule? (`NA * 0` is a tricky counterexample!)

R Code | Start Over | Hints | Run Code

```
# Si NA es finito, la multiplicación será 0, pero si es finito, la multiplicación
# será NaN, al no saber que es Na, no se puede saber si la multiplicación
# será NaN o 0, por lo tanto el resultado también es NA.
NA * 0
```

```
[1] NA
```

Previous Topic