

Functions

Writing your own functions

So far we've seen many functions, like `c()`, `class()`, `filter()`, `dim()` ...

Why create your own functions?

- Cut down on repetitive code (easier to fix things!)
- Organize code into manageable chunks
- Avoid running code unintentionally
- Use names that make sense to you

Writing your own functions

Here we will write a function that multiplies some number (x) by 2:

```
times_2 <- function(x) x * 2
```

When you run the line of code above, you make it ready to use (no output yet!).
Let's test it!

```
times_2(x = 10)
```

```
[1] 20
```

Writing your own functions: { }

Adding the curly brackets - {} - allows you to use functions spanning multiple lines:

```
times_2 <- function(x) {  
  x * 2  
}  
times_2(x = 10)
```

```
[1] 20
```

Writing your own functions: return

If we want something specific for the function's output, we use `return()`:

```
times_2 <- function(x) {  
  output <- x * 2  
  return(output)  
}  
times_2(x = 10)
```

```
[1] 20
```

Writing your own functions

Review: The syntax for a function is:

```
functionName <- function(inputs) {  
  <function body>  
  return(value)  
}
```

Writing your own functions: multiple inputs

Functions can take multiple inputs:

```
times_2_plus_y <- function(x, y) x * 2 + y  
times_2_plus_y(x = 10, y = 3)
```

```
[1] 23
```

Writing your own functions: defaults

Functions can have “default” arguments. This lets us use the function without using an argument later:

```
times_2_plus_y <- function(x = 10, y = 3) x * 2 + y  
times_2_plus_y()
```

```
[1] 23
```


Writing another simple function

Let's write a function, `sqdif`, that:

1. takes two numbers `x` and `y` with default values of 2 and 3.
2. takes the difference
3. squares this difference
4. then returns the final value

Writing another simple function

```
sqdif <- function(x = 2, y = 3) (x - y) ^ 2
```

```
sqdif()
```

```
[1] 1
```

```
sqdif(x = 10, y = 5)
```

```
[1] 25
```

```
sqdif(10, 5)
```

```
[1] 25
```

Writing your own functions: characters

Functions can have any kind of input. Here is a function with characters:

```
loud <- function(word) {  
  output <- rep( toupper(word), 5)  
  return(output)  
}  
loud(word = "hooray!")
```

```
[1] "HOORAY!" "HOORAY!" "HOORAY!" "HOORAY!" "HOORAY!"
```

Functions for tibbles

We can use `filter(row_number()==n)` to extract a row of a tibble:

```
cars <- read_kaggle()
```

```
get_row <- function(dat, row) dat %>% filter(row_number()==row)
```

```
get_row(dat = cars, row = 10)
```

```
# A tibble: 1 × 10
```

	RefId	IsBadBuy	PurchDate	Auction	VehYear	VehicleAge	Make	Model	Trim	SubModel
	<dbl>	<dbl>	<chr>	<chr>	<dbl>	<dbl>	<chr>	<chr>	<chr>	<chr>
1	10	0	12/7/2009	ADESA	2007	2	FORD	FIVE...	SEL	4D SEDA...

Functions for tibbles

`select(n)` will choose column `n`:

```
get_index <- function(dat, row, col){  
  dat %>% filter(row_number()==row) %>% select(col)  
}
```

```
get_index(dat = cars, row = 10, col = 8)
```

```
# A tibble: 1 × 1  
  Model  
  <chr>  
1 FIVE HUNDRED
```

Functions for tibbles

Including default values for arguments:

```
get_top <- function(dat, row = 1, col = 1){  
  dat %>% filter(row_number()==row) %>% select(col)  
}
```

```
get_top(dat = cars)
```

```
# A tibble: 1 × 1  
  RefId  
  <dbl>  
1      1
```

Using your custom functions: **sapply()**

Now that you've made a function... You can “apply” functions easily with `sapply()`!

These functions take the form:

```
sapply(<a vector or list>, some_function)
```

Using your custom functions: **sapply()**

□ There are no parentheses on the functions! □

```
sapply(cars, class)
```

RefId	IsBadBuy
"numeric"	"numeric"
PurchDate	Auction
"character"	"character"
VehYear	VehicleAge
"numeric"	"numeric"
Make	Model
"character"	"character"
Trim	SubModel
"character"	"character"
Color	Transmission
"character"	"character"
WheelTypeID	WheelType
"character"	"character"
VehOdo	Nationality
"numeric"	"character"
Size	TopThreeAmericanName
"character"	"character"

```
MMRAcquisitionAuctionAveragePrice
```

```
MMRAcquisitionAuctionCleanPrice
```


Using your custom functions “on the fly” to iterate

```
sapply(pull(cars, Veh0do), function(x) x / 1000)
```

[1]	89.046	93.593	73.807	65.617	69.367	81.054	65.328	65.805	49.921
[10]	84.872	80.080	75.419	79.315	71.254	74.722	72.132	80.736	75.156
[19]	65.925	84.498	54.586	66.536	98.130	59.789	65.663	52.106	88.958
[28]	76.173	65.393	80.064	77.694	56.300	78.241	57.723	78.434	82.944
[37]	76.304	55.711	76.586	65.078	65.403	86.889	68.990	80.949	52.774
[46]	72.191	59.858	79.576	73.291	50.227	82.146	58.024	40.919	87.643
[55]	80.968	50.308	80.795	62.239	87.008	64.060	77.677	58.888	63.557
[64]	90.026	89.705	64.511	75.513	80.608	95.558	35.796	83.501	70.148
[73]	76.052	72.479	84.542	61.081	86.483	43.898	57.338	59.425	79.957
[82]	78.559	48.386	80.117	65.795	51.145	88.366	55.909	86.702	81.424
[91]	65.379	74.954	49.328	73.810	43.412	78.412	74.026	64.822	80.491
[100]	85.003	65.711	56.064	62.230	62.190	67.426	75.806	88.991	89.849
[109]	81.338	80.077	77.233	66.681	82.526	81.930	74.131	72.417	64.118
[118]	71.423	64.650	85.388	95.443	69.337	46.563	84.905	71.062	80.999
[127]	66.545	67.785	71.952	70.741	94.318	69.440	54.268	59.072	86.028
[136]	64.677	68.874	64.554	73.988	23.881	50.532	60.554	91.558	63.377
[145]	59.391	44.367	44.515	83.238	92.532	68.165	87.775	86.414	36.142
[154]	80.788	93.346	73.963	68.183	64.839	75.484	59.287	63.151	46.695
[163]	58.897	65.363	75.237	85.042	87.701	92.816	97.221	73.726	47.550

Using your custom functions: `sapply()`.

```
cars_dbl <- cars %>% select(Make, Model, where(is.double))
```

```
Odo_updated <- sapply(pull(cars_dbl, VehOdo), times_2_plus_y)
```

```
cars_dbl %>%  
  mutate(Odo_2_y = Odo_updated) %>% select(c(1:2, 7:13))
```

```
# A tibble: 72,983 × 9
```

	Make	Model	VehOdo	BYRNO	VNZIP1	VehBCost	IsOnlineSale	WarrantyCost	Odo_2_y
	<chr>	<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1	MAZDA	MAZD...	89046	21973	33619	7100	0	1113	178095
2	DODGE	1500...	93593	19638	33619	7600	0	1053	187189
3	DODGE	STRA...	73807	19638	33619	4900	0	1389	147617
4	DODGE	NEON	65617	19638	33619	4100	0	630	131237
5	FORD	FOCUS	69367	19638	33619	4000	0	1020	138737
6	MITSUBI...	GALA...	81054	19638	33619	5600	0	594	162111
7	KIA	SPEC...	65328	19638	33619	4200	0	533	130659
8	FORD	TAUR...	65805	19638	33619	4500	0	825	131613
9	KIA	SPEC...	49921	21973	33619	5600	0	482	99845
10	FORD	FIVE...	84872	21973	33619	7700	0	1633	169747

```
# ... with 72,973 more rows
```

Applying functions with **across** from **dplyr**

`across()` makes it easy to apply the same transformation to multiple columns, allowing you to use `select()` semantics inside functions like `summarize()` and `mutate()`.

```
across( .cols = <columns>, .fns = function, ... )
```

- List columns first: `.cols =`
- List function next: `.fns =`
- Then list any arguments for the function

Applying functions with **across** from **dplyr**.

Combining with `summarize()`:

```
cars_dbl %>%  
  group_by(Make) %>%  
  summarize(across(.cols = everything(), .fns = mean))  
  
# A tibble: 33 × 12  
  Make      Model RefId IsBadBuy VehYear VehicleAge VehOdo  BYRNO VNZIP1 VehBCost  
  <chr>   <dbl> <dbl>   <dbl>   <dbl>      <dbl>  <dbl>  <dbl> <dbl>   <dbl>  
1 ACURA      NA 36021.   0.273   2003.      6.52 81732. 21851. 61217.   9039.  
2 BUICK       NA 35431.   0.157   2004.      5.65 76238. 19755. 51298.   6169.  
3 CADIL...    NA 34173.   0.152   2004.      5.24 73770. 20383. 50775.  10958.  
4 CHEVR...    NA 35417.   0.0975  2006.      3.97 73390. 26912. 58874.   6835.  
5 CHRYS...    NA 37614.   0.129   2006.      3.65 66814. 31268. 58562.   6507.  
6 DODGE       NA 36851.   0.103   2006.      3.75 68261. 36094. 58788.   7047.  
7 FORD        NA 36866.   0.154   2005.      4.75 76749. 19887. 59427.   6403.  
8 GMC         NA 35245.   0.116   2004.      5.61 79273. 18802. 58113.   8342.  
9 HONDA       NA 35109.   0.109   2004.      5.33 77877. 24161. 52659.   8350.  
10 HUMMER     NA 19533    0      2006      3    70809 21053 95673  11920  
# ... with 23 more rows, and 2 more variables: IsOnlineSale <dbl>,  
#   WarrantyCost <dbl>
```

Applying functions with **across** from **dplyr**.

Adding arguments to the function (`quantile()`) at the end:

```
cars_dbl %>%  
  group_by(Make) %>%  
  summarize(across(.cols = where(is.double), .fns = quantile, probs = 0.95))
```

A tibble: 33 × 11

	Make	RefId	IsBadBuy	VehYear	VehicleAge	VehOdo	BYRNO	VNZIP1	VehBCost
	<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1	ACURA	67522.	1	2005	8	93338.	36099.	92807	12093
2	BUICK	67803.	1	2007	8.05	95049.	52117	92337	8345.
3	CADILLAC	68611.	1	2006	7	87267.	34482.	85115.	11094
4	CHEVROLET	68895.	1	2008	7	92505	99750	94544	9170
5	CHRYSLER	69029.	1	2008	7	89784.	99761	92504	9280
6	DODGE	68446.	1	2008	7	91557.	99761	92504	10265
7	FORD	69731.	1	2007	8	95213.	52117	92807	9834
8	GMC	69012.	1	2006	8	94470	25100	92504	10912
9	HONDA	69827	1	2007	8	93811	99740	92504	10440
10	HUMMER	19533	0	2006	3	70809	21053	95673	11920

... with 23 more rows, and 2 more variables: IsOnlineSale <dbl>,
WarrantyCost <dbl>

Applying functions with **across** from **dplyr**.

Using different `tidyselect()` options:

```
cars_dbl %>%  
  group_by(Make) %>%  
  summarize(across(.cols = starts_with("Veh"), .fns = mean))
```

A tibble: 33 × 5

	Make	VehYear	VehicleAge	VehOdo	VehBCost
	<chr>	<dbl>	<dbl>	<dbl>	<dbl>
1	ACURA	2003.	6.52	81732.	9039.
2	BUICK	2004.	5.65	76238.	6169.
3	CADILLAC	2004.	5.24	73770.	10958.
4	CHEVROLET	2006.	3.97	73390.	6835.
5	CHRYSLER	2006.	3.65	66814.	6507.
6	DODGE	2006.	3.75	68261.	7047.
7	FORD	2005.	4.75	76749.	6403.
8	GMC	2004.	5.61	79273.	8342.
9	HONDA	2004.	5.33	77877.	8350.
10	HUMMER	2006	3	70809	11920

... with 23 more rows

Applying functions with **across** from **dplyr**.

Combining with `mutate()`:

```
cars_dbl %>%  
  mutate(across(.cols = starts_with("Veh"), .fns = round, digits = -3))
```

```
# A tibble: 72,983 × 12
```

	Make	Model	RefId	IsBadBuy	VehYear	VehicleAge	VehOdo	BYRNO	VNZIP1	VehBCost
	<chr>	<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1	MAZDA	MAZD...	1	0	2000	0	89000	21973	33619	7000
2	DODGE	1500...	2	0	2000	0	94000	19638	33619	8000
3	DODGE	STRA...	3	0	2000	0	74000	19638	33619	5000
4	DODGE	NEON	4	0	2000	0	66000	19638	33619	4000
5	FORD	FOCUS	5	0	2000	0	69000	19638	33619	4000
6	MITSUBI...	GALA...	6	0	2000	0	81000	19638	33619	6000
7	KIA	SPEC...	7	0	2000	0	65000	19638	33619	4000
8	FORD	TAUR...	8	0	2000	0	66000	19638	33619	4000
9	KIA	SPEC...	9	0	2000	0	50000	21973	33619	6000
10	FORD	FIVE...	10	0	2000	0	85000	21973	33619	8000

```
# ... with 72,973 more rows, and 2 more variables: IsOnlineSale <dbl>,
```

```
#   WarrantyCost <dbl>
```

Applying functions with `across` from `dplyr`.

Combining with `mutate()`:

```
cars_dbl %>%  
  mutate(across(.cols = everything(),  
                .fns = str_replace_all,  
                pattern = "A",  
                replacement = "a"))
```

```
# A tibble: 72,983 × 12
```

	Make	Model	RefId	IsBadBuy	VehYear	VehicleAge	VehOdo	BYRNO	VNZIP1	VehBCost
	<chr>	<chr>	<chr>	<chr>	<chr>	<chr>	<chr>	<chr>	<chr>	<chr>
1	MaZDa	MaZD...	1	0	2006	3	89046	21973	33619	7100
2	DODGE	1500...	2	0	2004	5	93593	19638	33619	7600
3	DODGE	STRa...	3	0	2005	4	73807	19638	33619	4900
4	DODGE	NEON	4	0	2004	5	65617	19638	33619	4100
5	FORD	FOCUS	5	0	2005	4	69367	19638	33619	4000
6	MITSUBI...	GaLa...	6	0	2004	5	81054	19638	33619	5600
7	KIa	SPEC...	7	0	2004	5	65328	19638	33619	4200
8	FORD	TaUR...	8	0	2005	4	65805	19638	33619	4500
9	KIa	SPEC...	9	0	2007	2	49921	21973	33619	5600
10	FORD	FIVE...	10	0	2007	2	84872	21973	33619	7700

```
# ... with 72,973 more rows, and 2 more variables: IsOnlineSale <chr>,
```

```
# WarrantyCost <chr>
```


Applying functions with `across` from `dplyr`.

Combining with `mutate()`:

```
# Child mortality data
```

```
mort <- read_mortality() %>% rename(country = `...1`)
```

```
mort %>%
```

```
  select(country, starts_with("194")) %>%
```

```
  mutate(across(.cols = c(`1943`, `1944`, `1945`),  
                .fns = replace_na,  
                replace = 0))
```

```
# A tibble: 197 × 11
```

	country	`1940`	`1941`	`1942`	`1943`	`1944`	`1945`	`1946`	`1947`	`1948`	`1949`
	<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1	Afghan...	NA	NA	NA	0	0	0	NA	NA	NA	NA
2	Albania	1.53	1.31	1.48	1.46	1.43	1.40	1.37	1.41	1.37	1.34
3	Algeria	NA	NA	NA	0	0	0	NA	NA	NA	NA
4	Angola	4.46	4.46	4.46	4.34	4.34	4.34	4.33	4.22	4.22	4.21
5	Argent...	0.641	0.603	0.602	0.558	0.551	0.510	0.503	0.496	0.494	0.492
6	Armenia	NA	NA	NA	0	0	0	NA	NA	NA	NA
7	Aruba	NA	NA	NA	0	0	0	NA	NA	NA	NA
8	Austra...	0.263	0.275	0.276	0.299	0.260	0.271	0.295	0.279	0.271	0.271
9	Austria	0.504	0.474	0.417	0.389	0.360	0.311	0.311	0.312	0.274	0.274

Website

Website