

# 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

The general 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
"character"	"character"
MMRAcquisitionRetailAveragePrice	MMRAcquisitonRetailCleanPrice
"character"	"character"
MMRCurrentAuctionAveragePrice	MMRCurrentAuctionCleanPrice
"character"	"character"
MMRCurrentRetailAveragePrice	MMRCurrentRetailCleanPrice



## Using your custom functions: `sapply()`

```
sapply(pull(cars, Veh0do), times_2_plus_y)
```

```
[1] 178095 187189 147617 131237 138737 162111 130659 131613 99845 169747
[11] 160163 150841 158633 142511 149447 144267 161475 150315 131853 168999
[21] 109175 133075 196263 119581 131329 104215 177919 152349 130789 160131
[31] 155391 112603 156485 115449 156871 165891 152611 111425 153175 130159
[41] 130809 173781 137983 161901 105551 144385 119719 159155 146585 100457
[51] 164295 116051 81841 175289 161939 100619 161593 124481 174019 128123
[61] 155357 117779 127117 180055 179413 129025 151029 161219 191119 71595
[71] 167005 140299 152107 144961 169087 122165 172969 87799 114679 118853
[81] 159917 157121 96775 160237 131593 102293 176735 111821 173407 162851
[91] 130761 149911 98659 147623 86827 156827 148055 129647 160985 170009
[101] 131425 112131 124463 124383 134855 151615 177985 179701 162679 160157
[111] 154469 133365 165055 163863 148265 144837 128239 142849 129303 170779
[121] 190889 138677 93129 169813 142127 162001 133093 135573 143907 141485
[131] 188639 138883 108539 118147 172059 129357 137751 129111 147979 47765
[141] 101067 121111 183119 126757 118785 88737 89033 166479 185067 136333
[151] 175553 172831 72287 161579 186695 147929 136369 129681 150971 118577
[161] 126305 93393 117797 130729 150477 170087 175405 185635 194445 147455
[171] 95103 126161 128131 176057 164331 169529 104229 99789 185567 92005
[181] 125983 157987 128919 121047 147453 142431 121063 133393 178063 72853
[191] 117649 145187 158033 177337 117001 190053 101291 177667 136083 116771
[201] 158571 161815 188025 173753 122641 158669 185797 119605 150219 135395
[211] 100773 116903 150143 147743 87073 111369 117365 125593 154359 138863
[221] 172935 138963 172691 158063 189015 155575 134863 146821 120405 98595
[231] 122633 168911 170527 145989 179541 137103 125913 150205 176349 105585
[241] 170671 138063 129113 132869 154103 168779 105735 91363 126995 157189
[251] 186793 106929 166165 146321 150317 147035 130921 185587 164545 177449
[261] 144043 154763 101549 117861 159799 183549 124545 133139 126161 164219
[271] 120683 185951 149943 124469 158165 111515 107287 147665 177059 188145
```

# Using your custom functions “on the fly” to iterate

```
apply(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
[172]	63.079	64.064	88.027	82.164	84.763	52.113	49.893	92.782	46.001
[181]	62.990	78.992	64.458	60.522	73.725	71.214	60.530	66.695	89.030
[190]	36.425	58.823	72.592	79.015	88.667	58.499	95.025	50.644	88.832
[199]	68.040	58.384	79.284	80.906	94.011	86.875	61.319	79.333	92.897
[208]	59.801	75.108	67.696	50.385	58.450	75.070	73.870	43.535	55.683
[217]	58.681	62.795	77.178	69.430	86.466	69.480	86.344	79.030	94.506
[226]	77.786	67.430	73.409	60.201	49.296	61.315	84.454	85.262	72.993
[235]	89.769	68.550	62.955	75.101	88.173	52.791	85.334	69.030	64.555
[244]	66.433	77.050	84.388	52.866	45.680	63.496	78.593	93.395	53.463

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

`across()` makes it easy to apply the same transformation to multiple columns. Usually used with `summarize()`.

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

```
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	CHRY...	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**.

Combining with `summarize()`:

```
# Adding arguments to the end!
```

```
#
```

```
cars_dbl %>%
```

```
  group_by(Make) %>%
```

```
  summarize(across(.cols = everything(), .fns = mean, na.rm = TRUE))
```

```
# 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	CHRY...	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**.

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 **dp1yr**.

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
10	Azerba...	NA	NA	NA	0	0	0	NA	NA	NA	NA

```
# ... with 187 more rows
```

## Summary

- Simple functions take the form:
  - `NEW_FUNCTION <- function(x, y) x + y ..`
  - Can specify defaults like `function(x = 1, y = 2)`
- Apply your functions with `sapply(<a vector or list>, some_function)`
- Use `across()` to apply functions across multiple columns of data

# Website

- ▮ [Class Website](#)

- ▮ [Lab](#)

The end!