```
library(tidyverse)
```

```
── Attaching packages ─────────────────────────────────────── tidyverse 1.3.0 ──
✓ ggplot2 3.3.3     ✓ purrr   0.3.4
✓ tibble  3.0.6     ✓ dplyr   1.0.4
✓ tidyr   1.1.2     ✓ stringr 1.4.0
✓ readr   1.4.0     ✓ forcats 0.5.1
── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
✖ dplyr::filter() masks stats::filter()
✖ dplyr::lag()    masks stats::lag()
```

The pipe

We've used the pipe to simplify our code frequently in this course. Let's take a closer look at this operator.

- The pipe comes from `magrittr` package, which is part of the tidyverse suite.
- The pipe takes the output from the line and "pipes" it to the next line as the first argument of the next function.

```
mpg %>%
    filter(class == "compact") %>%
    print()

# equivalent to:
print(filter(mpg, class == "compact"))
```

A tibble: 47 × 11

manufacturer	model	displ	year	cyl	trans	drv	city	hwy	fl	class
<chr>	<chr>	<dbl>	<int>	<int>	<chr>	<chr>	<int>	<int>	<chr>	<chr>
audi	a4	1.8	1999	4	auto(l5)	f	18	29	p	compact
audi	a4	1.8	1999	4	manual(m5)	f	21	29	p	compact
audi	a4	2.0	2008	4	manual(m6)	f	20	31	p	compact
audi	a4	2.0	2008	4	auto(av)	f	21	30	p	compact
audi	a4	2.8	1999	6	auto(l5)	f	16	26	p	compact
audi	a4	2.8	1999	6	manual(m5)	f	18	26	p	compact
audi	a4	3.1	2008	6	auto(av)	f	18	27	p	compact
audi	a4 quattro	1.8	1999	4	manual(m5)	4	18	26	p	compact
audi	a4 quattro	1.8	1999	4	auto(l5)	4	16	25	p	compact
audi	a4 quattro	2.0	2008	4	manual(m6)	4	20	28	p	compact
audi	a4 quattro	2.0	2008	4	auto(s6)	4	19	27	p	compact
audi	a4 quattro	2.8	1999	6	auto(l5)	4	15	25	p	compact
audi	a4 quattro	2.8	1999	6	manual(m5)	4	17	25	p	compact
audi	a4 quattro	3.1	2008	6	auto(s6)	4	17	25	p	compact
audi	a4 quattro	3.1	2008	6	manual(m6)	4	15	25	p	compact
nissan	altima	2.4	1999	4	manual(m5)	f	21	29	r	compact
nissan	altima	2.4	1999	4	auto(l4)	f	19	27	r	compact
subaru	impreza awd	2.5	2008	4	auto(l4)	4	20	25	r	compact
subaru	impreza awd	2.5	2008	4	auto(s4)	4	20	27	r	compact
subaru	impreza awd	2.5	2008	4	manual(m5)	4	19	25	p	compact
subaru	impreza awd	2.5	2008	4	manual(m5)	4	20	27	r	compact
toyota	camry solara	2.2	1999	4	auto(l4)	f	21	27	r	compact
toyota	camry solara	2.2	1999	4	manual(m5)	f	21	29	r	compact
toyota	camry solara	2.4	2008	4	manual(m5)	f	21	31	r	compact
toyota	camry solara	2.4	2008	4	auto(s5)	f	22	31	r	compact
toyota	camry solara	3.0	1999	6	auto(l4)	f	18	26	r	compact
toyota	camry solara	3.0	1999	6	manual(m5)	f	18	26	r	compact
toyota	camry solara	3.3	2008	6	auto(s5)	f	18	27	r	compact
toyota	corolla	1.8	1999	4	auto(l3)	f	24	30	r	compact
toyota	corolla	1.8	1999	4	auto(l4)	f	24	33	r	compact
toyota	corolla	1.8	1999	4	manual(m5)	f	26	35	r	compact
toyota	corolla	1.8	2008	4	manual(m6)	f	28	37	r	compact
toyota	corolla	1.8	2008	4	auto(l4)	f	26	35	r	compact
volkswagen	gti	2.0	1999	4	manual(m5)	f	21	29	r	compact
volkswagen	gti	2.0	1999	4	auto(l4)	f	19	26	r	compact
volkswagen	gti	2.0	2008	4	manual(m6)	f	21	29	r	compact
volkswagen	gti	2.0	2008	4	auto(s6)	f	22	29	p	compact
volkswagen	gti	2.8	1999	6	manual(m5)	f	17	24	r	compact
volkswagen	jetta	1.9	1999	4	manual(m5)	f	33	44	d	compact
volkswagen	jetta	2.0	1999	4	manual(m5)	f	21	29	r	compact
volkswagen	jetta	2.0	1999	4	auto(l4)	f	19	26	r	compact
volkswagen	jetta	2.0	2008	4	auto(s6)	f	22	29	p	compact
volkswagen	jetta	2.0	2008	4	manual(m6)	f	21	29	p	compact
volkswagen	jetta	2.5	2008	5	auto(s6)	f	21	29	r	compact
volkswagen	jetta	2.5	2008	5	manual(m5)	f	21	29	r	compact
volkswagen	jetta	2.8	1999	6	auto(l4)	f	16	23	r	compact
volkswagen	jetta	2.8	1999	6	manual(m5)	f	17	24	r	compact

```
sum(c(1,2,3,4))

c(1,2,3,4) %>%
    sum()
```

10
10

Why use the pipe?

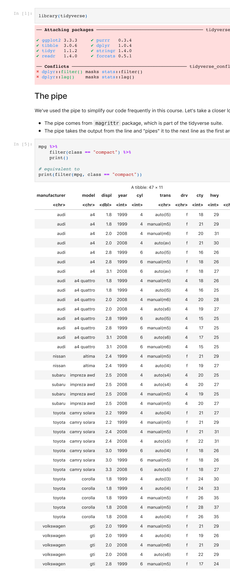
- Transform an object in a series of steps that is easy to debug and easy for a human to read

```
# four ways to compute the mean of the unique values

# intermediate assignments
x <- c(1,2,3,4,1,2,3,4)
y <- unique(x)
(z <- mean(y))

# overwrite the same variable
x <- c(1,2,3,4,1,2,3,4)
x <- unique(x)
(x <- mean(x))

# compose all the function
(z <- mean(unique(c(1,2,3,4,1,2,3,4))))


# use the pipe
(x <- c(1,2,3,4,1,2,3,4) %>%
   unique() %>%
   mean())
```

2.5
2.5
2.5
2.5

Caution!

```
# Once we have many lines, even the pipe can be hard to read
mpg %>%
    filter(cty >= 20) %>%
    mutate(combined_mpg = cty + hwy) %>%
    group_by(class) %>%
    summarize(mean_combined_mpg = mean(combined_mpg)) %>%
    select(mean_combined_mpg, everything()) %>%
    ggplot(aes(x = class, y = mean_combined_mpg)) +
    geom_point()
```




```
# an intermediate variable here might improve readability
mean_mpg <- mpg %>%
              filter(cty >= 20) %>%
              mutate(combined_mpg = cty + hwy) %>%
              group_by(class) %>%
              summarize(mean_combined_mpg = mean(combined_mpg)) %>%
              select(mean_combined_mpg, everything())

ggplot(mean_mpg, aes(x = class, y = mean_combined_mpg)) +
    geom_point()
```



```
# if we have multiple objects being manipulated and combined, pipe may not be the right tool for each line

mean_mpg <- mpg %>%
              mutate(combined_mpg = cty + hwy) %>%
              group_by(class) %>%
              summarize(mean_combined_mpg = mean(combined_mpg))

sum_mpg <- mpg %>%
              mutate(combined_mpg = cty + hwy) %>%
              group_by(class) %>%
              summarize(sum_combined_mpg = sum(combined_mpg))

full_join(mean_mpg, sum_mpg)
```

```
Joining, by = "class"
```

A tibble: 7 × 3

class	mean_combined_mpg	sum_combined_mpg
<chr>	<dbl>	<int>
2seater	40.20000	201
compact	48.43553	2276
midsize	46.04878	1888
minivan	38.18182	420
pickup	29.87879	986
subcompact	48.51429	1698
suv	31.62903	1961

The **T-pipe**, `%T>%`, pipes the output of the line to the following two lines.

```
mpg %>%
    mutate(combined_mpg = cty + hwy) %>%
    group_by(class) %>%
    summarize(mean_combined_mpg = mean(combined_mpg)) %T>%
    print() %>%
    ggplot(aes(x = class, y = mean_combined_mpg)) + geom_point()
```

Functions in R

- When you find yourself "copy-pasting" the same code over and over again or applying the same steps to similar objects, it might be time to write a custom function.
  - Copy-pasting can create errors
  - If you want to edit your procedure, you only have to do it once
  - Hide ugly, messy, or non-descriptive code for other users, **clear name**
- At its heart, a function is an input/output machine. You need to know:
  - The inputs (or **arguments**)
  - What to do to the inputs (or **body**)
  - What you want as an output
  - *A good name*

```
# add two numbers
add_two <- function(x, y) {
    x+y
}
# the last line here is the output of the function
```

```
add_two(1,5)
```

6

```
# one way to think about the above function
x <- 1
y <- 5
x+y
```

6

Sometimes the curly brackets are omitted, but only do this for **very** short functions

```
add_two <- function(x,y) x+y
```

```
add_two(1,5)
```

6

Naming your function:

- Just as important as naming your variables (or more!)
- Short is better, but clear is the most important

Examples:

- `f()` is too short
- `the_best_function_ever()` is not clear
- `collapse_years()` or `collapseYears()` but **not** both
- `mean()` is better than `compute_mean()`
  - be careful of using "compute","find","get", etc.

```
# Don't do this!
T <- FALSE   # T is already TRUE
c <- 10      # c() is used for vectors (can be confusing)
mean <- function(x) sum(x)   # don't overwrite base R functions when possible
```

Attempt:

- Write a function that takes as input a numeric vector and outputs the vector obtained from subtracting the mean from each element.
- Write a function that takes as input a tibble and outputs a tibble of the first 3 columns and the first 10 rows.
- Verify your function works as expected!
- Hint: Make sure your body works before writing a function.

```
# do this before writing add_two to make sure
x <- 1
y <- 2
x+y
```

3

```
substract_mean <- function(x) {
    y <- mean(x)
    x - y
}

# VERIFY
substract_mean(c(1,1,1,1))
```

0 - 0 - 0 - 0

```
# hint for second functions head(tibble, n) and select()
```

Conditional Statements

- if (if..then)
- if...else
- if...else if...else

if...then

`if (condition) { do this }`

- `condition`: here is either `TRUE` or `FALSE`
- Brackets can be omitted if it fits nicely on one line (like with functions above)

```
x <- 3

if (x > 1) {
    print("Number is greater than one")
}
```

[1] "Number is greater than one"

if...else....

`if (condition) {do this} else {do that}`

```
x <- 0

if (x > 1) {
    print("Number is greater than one")
} else {      # the "else" has to be on the same line as the end curly bracket
    print("Number is less than or equal to one")
}
```

[1] "Number is less than or equal to one"

Can also use a vectorized version:

`ifelse(condition, do this, do that)`

```
ifelse(x > 1, ">1", "<=1")
```

"<=1"

Attempt

- Write a function that determines whether an integer is even or odd.

```
# mod function (7 mod 2)
7 %% 2
```

1

```
x <- 2

if (x %% 2 == 0) {
    print("Even")
} else {      # the "else" has to be on the same line as the end curly bracket
    print("Odd")
}
```

[1] "Even"